USENIX

Proceedings of LISA '05: 19th Large Installation System Administration Conference

San Diego, CA, USA, Dec. 4–9, 2005

conference

proceedings

LISA '05
19th Large Installation System Administration Conference

San Diego, California, USA

December 4–9, 2005

Sponsored by

The USENIX Association and **SAGE**

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

SAGE

The People Who Make IT Work

For additional copies of these proceedings contact

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Telephone: +1 510-528-8649
http://www.usenix.org
<office@usenix.org>

Past LISA Conferences

LISA '04: 18th Large Installation System Administration Conference	Nov. 14-19, 2004	Atlanta, GA
LISA '03: 17th Large Installation Systems Administration Conference	Oct. 26-31, 2003	San Diego, CA
LISA '02: 16th Systems Administration Conference	Nov. 3-8, 2002	Philadelphia, PA
LISA 2001: 15th Systems Administration Conference	Dec. 2-7, 2001	San Diego, CA
LISA 2000: 14th Systems Administration Conference	Dec. 3-8, 2000	New Orleans, LA
LISA '99: 13th Systems Administration Conference	Nov. 7-12, 1999	Seattle, WA
LISA '98: 12th Systems Administration Conference	Dec. 6-11, 1998	Boston, MA
LISA '97: 11th Systems Administration Conference	Oct. 26-31, 1997	San Diego, CA
LISA '96: 10th System Administration Conference	Sept. 29-Oct. 4, 1996	Chicago, IL
LISA '95: 9th System Administration Conference	Sept. 18-22, 1995	Monterey, CA
LISA '94: 8th USENIX System Administration Conference	Sept. 19-23, 1994	San Diego, CA
LISA '93: USENIX 7th System Administration Conference	Nov. 1-5, 1993	Monterey, CA
LISA VI: 6th Systems Administration Conference	Oct. 19-23, 1992	Long Beach, CA
5th Large Installation Systems Administration Conference	Sept. 30-Oct. 3, 1991	San Diego, CA
4th Large Installation System Administrator's Conference	Oct. 17-19, 1990	Colorado Springs, CO
Workshop on Large Installation Systems Administration III	Sept. 7-8, 1989	Austin, TX
Workshop on Large Installation Systems Administration	Nov. 17-18, 1988	Monterey, CA
Workshop on Large Installation Systems Administration	April 9-10, 1987	Philadelphia, PA

ISBN 1-931971-38-2

USENIX Association

Proceedings of the 19th Large Installation System Administration Conference (LISA '05)

December 4-9, 2005
San Diego, CA, USA

ACKNOWLEDGMENTS

PROGRAM CHAIR
David N. Blank-Edelman, *Northeastern University CCIS*

PROGRAM COMMITTEE
Gerald Carter, *Samba Team/Hewlett-Packard*
Strata Rose Chalup, *VirtualNet Consulting*
Lee Damon, *University of Washington*
Rudi van Drunen, *Leiden Cytology and Pathology Labs*
Joe Gross, *Independent Consultant*
Tom Limoncelli, *Cibernet Corp.*
John "Rowan" Littell, *Earlham College*
Tom Perrine, *Sony Computer Entertainment America*
Yi-Min Wang, *Microsoft Research*
David Williamson, *Tellme Networks*
Elizabeth Zwicky, *Acuitus*

INVITED TALKS COORDINATORS
Adam S. Moskowitz, *Menlo Computing*
William LeFebvre, *Independent Consultant*

GURU IS IN COORDINATOR
Philip Kizer, *Texas A&M University*

WORK-IN-PROGRESS REPORTS COORDINATOR
Esther Filderman, *Pittsburgh Supercomputing Center*

PEER INTERACTION COORDINATORS
Joe and Lorah Gross

WORKSHOP OUTREACH
Bob Apthorpe, *St. Edward's University*

EXPERIMENTAL NETWORK TEAM LEADER
David Nolan, *Carnegie Mellon University*

NATIVE GUIDE
Tom Perrine, *Sony Computer Entertainment America*

WORKSHOPS COORDINATOR
Luke Kanies, *Reductive Consulting*

MANAGING SYSADMINS WORKSHOP
Tom Limoncelli, *Cibernet Corp.*
Cat Okita, *Hewlett-Packard*

CLUSTERS WORKSHOP
Susan Coghlan, *Argonne National Laboratory*
Narayan Desai, *Argonne National Laboratory*

CONFIGURATION MANAGEMENT WORKSHOP
Paul Anderson, *University of Edinburgh*

UNIVERSITY ISSUES WORKSHOP
David Parter, *University of Wisconsin*
Philip Kizer, *Texas A&M University*
John "Rowan" Littell, *Earlham College*

USING THE NEW SOCIAL TECHNOLOGIES ... WORKSHOP
Strata R. Chalup, *Virtual.Net Inc.*

ADVANCED TOPICS WORKSHOP
Adam Moskowitz, *Menlo Computing*

AFS WORKSHOP
Esther Filderman, *The OpenAFS Project*

SYSTEM ADMINISTRATION EDUCATION WORKSHOP
Steven Jenkins, *East Tennessee State University*

PROCEEDINGS TYPESETTING
Rob Kolstad, *USENIX*

CONFERENCE ORGANIZATION
The USENIX Association Staff

EXTERNAL REVIEWERS

Steven Alexander
Bob Apthorpe
Kenytt Avery
Michael Ciavarella
Alva Couch
Jon Finke
Aurilee V. Gamboa
Paul Guglielmino
David Harnick-Shapiro
Robert Haskins
Jason Heiss
Ray Hiltbrand
Nathan Hruby
Tim Hunter
Steven Jenkins
Mark R. Lindsey
Shane B. Milburn
Mario Obejas
Cat Okita
Scott Orr
Kathryn Penn
Taiss Quartapa
Mark D. Roth
Nathan Schimke
Benjamin Smith
Len Smith
Chad Verbowski
Pat Wilson

CONTENTS

FRIDAY, DECEMBER 9

INDEX OF AUTHORS

Message from the Program Chair

Dear Reader:

It took me a long time to find this conference. Even though I had already been working in the field for about 10 years, it was at my first LISA conference that I learned there was an entire world of system administration beyond my office – a world of research challenges, best practices, and a whole set of peers grappling with the same challenges.

Now, many years after my first LISA, it is my pleasure as Program Chair to welcome you – or welcome you back – to this community.

You have in your hand, on your screen (or in your implant, hello Future!) the full text of the papers accepted for presentation in the refereed papers track of the 19th LISA conference. It is one of the places where the history of our field is being written. Dedicated and talented people labored this year to create this body of work. Submission authors, program committee members, and external reviewers all devoted kerjillions of hours to the advancement of our field. I'm very grateful for their efforts and I know future generations of sysadmins will be as well.

Similar valiant levels of effort have been expended by the Invited Talks coordinators (Adam and Bill), the coordinators for the various tracks and activities (Dan, Esther, Joe, Lorah, Luke, and Philip), and all of the other volunteers. Thanks are due to Rob who put the final polish on each paper until it gleamed bright enough to include in the collection in front of you now. I've also been very lucky to work with the fabulous USENIX staff, who help keep the train on the tracks no matter how the Program Chair drives. If you have appreciated this conference, please be sure to thank all of the LISA organizers/volunteers and the USENIX staff for their efforts.

I have often thought that somebody should write an unabashed love letter to the sysadmin community. As I enter my twentieth year as a sysadmin, my work on this conference, the 19th LISA to date, is the closest I could come to such a thing. I hope this year's LISA conference is as fabulous for you as my first LISA experience was for me. I look forward to joining you at many LISA conferences to come.

Respectfully and with deep affection,
David N. Blank-Edelman
LISA '05 Program Chair

GULP: A Unified Logging Architecture for Authentication Data

Matt Selsky and Daniel Medina – Columbia University

ABSTRACT

We have implemented the Grand Unified Logging Project, GULP, a flexible aggregation system for authentication log data. The system merges disparate logs stored across various servers into a single format according to an XML schema. This single format is logged to a database and queried via a web interface. The strength of this system lies in the ability to correlate information across multiple logging sources and display relevant information through a simple interface.

Introduction

At Columbia University, each person is given a unique username or "UNI" (University Network ID) by the Academic Information Systems (AcIS) group. A process is in place for UNIs to be activated, creating a password that allows access to various services.

These services run on many different hosts and have disparate logging facilities. For example, when a student logs into CourseWorks (web-based course management), a successful authentication record is stored in the CourseWorks database. Other logins, to CubMail (web-based email), CUNIX and PINEX (shell servers), and elsewhere, are similarly logged, but to other locations on local disk [2].

Logging data is stored in a variety of formats and is typically stored locally on the host which provides the service. Some example services and the logs formats they use:

UNIX wtmpx records (remote login to a server), unpacked from the binary format [5]:

```
dnm17 pts/42 mutie.cc.columbia.edu
       Fri Oct 29 09:21 - 10:08 (00:47)
```

Secure web servers run Apache and log in the common Apache text format [6]; see Display 1.

WIND (Web Identification Network Daemon) [3], which provides sign-on to various web applications, has a custom text format:

```
2004-10-29 09:21:00,000 Login - success
for dnm17:switchmgr (128.59.31.101)
[pass:...:...] r
```

CubMail is a webmail client based on the Horde's IMP project [4]. Logs are text-based; see Display 2.

We could send the logs via syslog to a remote host, but we currently have it configured to only log locally. Processing these logs is not as intensive as is seen in commercial enterprises, but our logs are still quite sizeable [15]. During the summer months alone, there are about 10 GB/week of logs from the main webservers, and another 9 GB/week from the mail servers. Early in the school year we have observed 15.2 GB/week and 12.5 GB/week for web and mail, respectively.

Table 1 on the next page lists the main services we provide, along with the methods they use for logging authentication events.

Problem

Web and other logs are already harvested for usage statistics. Typically, there is an operational need to determine the number of users of a service, the applications being used to access services (browser client, email client, etc.), and so on. These statistics are collected periodically, usually a few times each semester.

The authentication information contained in these logs is valuable for a variety of other purposes. We generate usage reports for budget requests and capacity planning, demographic reports to show which university divisions are using our services (and therefore which schools should purchase account upgrades for all their students), and client software reports to ascertain the software people are using to access our services. This helps us to determine utilization of site-licensed software and also to plan software to support in the future.

```
mutie.cc.columbia.edu - dnm17 [29/Oct/2004:09:21:00 -0500]
"(GET /sec/acis/networks/index.html HTTP/1.1)"
200 202573 "(ref -)" " (client Mozilla/5.0 (Macintosh; U; PPC
Mac OS X Mach-O; rv:1.7.3) Gecko/20040911 Firefox/0.10)"
```

Display 1: Common Apache text format log entry.

```
Oct 29 09:21:00 HORDE [notice] [imp] Login success for
dnm17@columbia.edu [128.59.31.101] to {localhost:143}
[on line 92 of "/etc/httpd/htdocs/horde/imp/redirect.php"]
```

Display 2: A custom text-based log entry from CubMail.

Authentication information (UNI and password) gives access to an individual's personal information, including payroll, financial aid data, grades and course registration, email, and personal contact information.

We use Kerberos [1] to provide a centralized authentication mechanism, but Kerberos logs lack interesting features (since the transactions are brokered between the Kerberos server and the service or application server). We wish to preserve as much client information relevant to the authentication as possible. Several of our services do not natively support Kerberos so they request a ticket on behalf of the user using the plaintext password. This type of event shows up in the Kerberos logs without any information about the end-host or any indication that the authentication attempt was successful; the log only shows that an initial ticket request was made from the service host for the user.

The AcIS security group regularly receives requests regarding personal login information from persons who believe their accounts are compromised. A decade ago, before the explosion in web-based applications, it would suffice to direct users to the onall [7] command, which in conjunction with the last command, would show the most recent logins on all of the UNIX timeshare hosts. last logs were never centralized as in [17], but the onall utility made searching for these standard logs easier.

The security group also receives requests to determine the owner of a particular host (usually in the form of an Internet Protocol (IP) address). Due to the model of local network access at Columbia (so-called "free love"), users are not required to log into the network to use it [8]. IP addresses may be linked to users via a multi-stage network logging procedure (mapping IP addresses to hardware addresses to switch ports to room information to room registration), or via authentication information. The latter is preferable when available.

A recent problem for the security group has been various applications (sometimes called "SpyWare") that proxy user web traffic. Some of these applications proxy not just normal web traffic, but also SSL-protected (HTTPS) traffic.[1] This is equivalent to the user sending their UNI and password information to a third party (along with the data contained in the pages she is visiting).

Since the Kerberos logs are not suitable for harvesting the authentication data we want, we must look to the logs stored on the individual servers, as described above. However these logs are not easily searchable through any standard interface.

Too often, the AcIS security group finds itself in a reactive position, responding to incidents that have become operational threats (such as mass account compromise, break-ins, and malware epidemics). During these incidents, there is typically a slow gathering of available information that may take hours and involve numerous staff members. Once logging information is centralized, we can gather this information much more rapidly. The application of data mining techniques may even enable proactive operations in the face of emerging threats.

Solution

We have implemented a flexible aggregation system that allows for easy querying of the relevant authentication data from disparate log sources. We gave the solution the moniker "GULP" (Grand Unified Logging Project). This was preferable to the half-serious "TIA" (Tracking ID Access/Total Information Awareness) and "ECHELON" (Experimentally Centralizing Host's Every Log-On Name). We examined several mechanisms for extracting useful features from this data.

Centralized Logging

To centralize the authentication information we want, we transform the log files to an XML file that is described by an XML schema. This document includes only the "interesting" information that we have defined, reducing the total amount of data retained.

Centralized Searching

This single format is then searched for useful information via a web interface. The advantage of this system lies in the ability to correlate information across multiple logging sources easily. Advanced searches can be defined and saved for future re-evaluation.

Data Mining

With the limited features we have from our authentication data, we can extract information regarding abnormal behavior. For example, evidence of spyware or a proxy server would be many different users connecting from a single source or network (off the campus network). A login from a source not seen before may indicate unauthorized access to an account. The ultimate goal, however, would be to provide a system to allow a

[1] By installing a trusted root certificate [9].

Service	*Function*	*Logging*
CUNIX	Shell servers (General purpose)	local, wtmpx
PINEX	Shell-based E-mail	local, wtmpx
CubMail	Web-based E-mail	local, custom
CourseWorks	Course-related materials	remote DB
Secure Web	SSL-protected pages on www1	local, Apache
WIND	Web-app sign-on platform	local, custom
RADIUS	VPN and dialup authentication	local, RADIUS detail

Table 1: Main services provided.

member of the security group to create rules as needed, rather than using hard-coded signatures.

Implementation

Centralized Logging

The advantage to using XML is that we can publish our schema and leave the responsibility for extracting the relevant data from the logs to the maintainer of the application generating the logs. The maintainer can then validate the generated document before contributing it to the central repository.[2] While we did not interact with any outside parties for the purposes of this project, it is easy to perform the required validation. The schema used is included in the Appendix.

The XML representations of the log files may then be transferred and stored in a relational database (MySQL, in our case) [10].

For our project, we used logs from UNIX timeshare hosts (CUNIX and PINEX), web application log-in servers (WIND), secure web servers (WWWS), and webmail (CubMail), all of which we have described above. We chose the timeshare hosts because "last" logs are traditionally important sources of remote login information (although fewer users now log in directly). We chose WIND logs because it controls access to some of the most important web-based applications at the university, including payroll information. We chose secure web logs because the format is very common and numerous proxies can be found and accessed via the secure web servers. We chose CubMail because it is a popular application, used by approximately 60% of our users.

We created simple parsers for each of the logs we intended to use. In the case of the "last", Apache, and Horde log parsers, we hope to have decent reuse potential, while WIND logs will probably be unique to our site.

Writing the parsers is fairly simple, with most parsers being less than 150 lines of code (and much of the code just setting up the connection to the database). The difficult parts of the parser are writing the regular expressions to extract revelant data from each log entry in the log file, dealing with disparate date and time formats, and reading binary log data. The more complex parsers also need to attempt to re-create session information using login and logout records that may not match up. We do not currently attempt to canonicalize usernames to the UNI since we do this in the web interface, by searching for the UNI and any other usernames associated with a person.

Centralized Searching

We created a simple web form, protected by an .htaccess file restricting access to our Security group, that allows searching via a username or remote IP address. The information returned includes the remote hostname, the service used, the local server that exported the log, the start and end time of the session (only the start time where the session concept does not apply), and a note if applicable (such as the TTY used, or the web page or service accessed). A link to an external WHOIS site is included for more information about the remote host [11].

The username and remote address are also linked back to the CGI to allow easy inversion of the search on either term. This type of inversion is quite typical (when, say, trying to determine what other logins have come from a strange address). In this manner, we have slightly more features than a simple log-grepper allows.

We have also created an additional web form to show users their own authentication history, after logging in, but we have not deployed it as of yet.

Data Mining

We had several pre-conceptions about what would constitute an anomalous login. We conjectured that there would be two kinds of (global) abnormalities: many connections coming from a single address for many different users; and a single user logging in from many different locations.

We collected frequency statistics for both of the above abnormalities and quickly discovered that our assumptions were not refined enough. Of 40,000 user accounts observed over three weeks, over 10,000 were seen from more than six remote locations. Almost 6,000 were seen logging in from over 10 locations. (10 users from a German ISP logged in from more than 60 distinct locations within that ISP's address space).

We attempted to use BGP [12] information to retrieve the Autonomous System (AS) number associated with each remote IP address. In this case, a user's 60 remote locations would be represented by a single AS number (belonging to the ISP). According to these new metrics, we find that most users typically log in from fewer than three ASs (work, home, and the Columbia network). Nevertheless, with 40,000 users, we will still experience many false alerts (and miss many legitimate violations).

In the other direction, we found almost 900 addresses (of over 120,000) from which more than 12 users logged into our systems. Of these, we know that some are classified by our security group as malicious proxies (as described above). Many of the remaining addresses belong to benign web proxies, NAT-ing routers, and corporate gateways.

Applications

In the end, the search tool is more useful for revealing anomalous behavior than a global set of rules. Allowing limited access and providing a useful interface, we can deploy the search tool to the larger

[2] Experience has shown that publishing a required schema that cannot be easily validated by the source and repository parties is pointless.

user community. Colorizing logins from various sources by network, a user may easily audit her own login history [13, 16].

A user is more likely to be aware of what qualifies as an anomalous login than a system administrator responsible for thousands of users (especially given our diverse user population and the limit of the log records we are processing). Making this record available to the user is no different than a phone bill or credit card bill, an itemized list that the user can use to check for fraudulent activity and transactions. We show an example of this report in Figure 1 below.

We used this user search tool to investigate twelve recent security incidents reported by end-users. One such incident involved a student travelling overseas; he had used a computer terminal "administered by a guy who admitted to me in a moment of intoxication that he's a criminal hacker." Needless to say, the student was concerned about the security of his account.

In six of the twelve cases, the tool confirmed the suspicions of the user that someone else was using their account. In the other cases, we did not observe any anomalous patterns, possibly due to either gaps in our data or gaps in our coverage (we are not yet collecting data from all available log sources). In one instance we identified a supposedly secure web application on a departmental server that was in fact using plaintext ftp for file uploads to CUNIX.

In another incident, the identity of the miscreant was discovered. A student suspected that someone was reading her email because she often found her message flags altered. Using the search tool, the security group found a number of abnormal logins from a public campus terminal. Inverting the search on the public terminal, they found that the same individual had logged in to the terminal before the complaining student. Apparently, the miscreant would check his mail first, then hers.

The security group can also create custom searches as required. The search shown in Figure 2 below quickly identifies all malicious MarketScore proxies (as defined above). When these proxies were first identified, it took two days to formulate the entire list of proxy sources and countless staff time since different staff members were familiar with the log locations and contents for different services. Currently, logs are collected of users of the proxies once a week. With this tool, a current list can be created instantly.

Future Work

Numerous areas for development are open to us now that we have a viable central logging system. We also see a number of improvements that can be made to the applications we have already created.

We will take steps to properly normalize the username (certain logs do not record the UNI of an individual and instead log a username, which in the case of staff members, may not be equivalent). We have currently handled this in the web form.

We will further improve the idea of a "session" by correlating login and logout messages from some of the sources that did not clearly identify records as belonging to a particular session (as has been done with CubMail logins).

We will expand the logs that we feed into the system, including POP, IMAP, authenticated SMTP, RADIUS, and CourseWorks logs.

medina	160.39.246.251	dyn-wireless-246-251.dyn.columbia.edu	cunix	walnut	2004-11-29 17:11:25	2004-11-29 17:22:19
medina	160.39.246.251	dyn-wireless-246-251.dyn.columbia.edu	cunix	banana	2004-11-29 18:09:58	2004-11-30 01:24:30
medina	70.19.109.194	pool-70-19-109-194.ny325.east.verizon.net	cunix	papaya	2004-11-29 23:50:55	2004-11-30 00:28:36
medina	70.19.109.194	pool-70-19-109-194.ny325.east.verizon.net	cunix	mango	2004-11-30 07:54:22	2004-11-30 09:09:55
medina	128.59.25.155	dynamic-25-155.dyn.columbia.edu	cunix	mango	2004-11-30 10:46:48	2004-11-30 12:57:39
medina	128.59.31.101	mutie.cc.columbia.edu	cunix	hazelnut	2004-11-30 13:04:35	2004-11-30 17:12:46
medina	128.59.59.215	manheru.cc.columbia.edu	pinex	persimmon	2004-11-30 22:11:45	2004-11-30 22:16:02

Figure 1: Sample search on user medina.

user2111	216.148.246.70	proxys.sj3.marketscore.com	CubMail	jujube	2004-12-06 01:05:04
user2111	216.148.246.70	proxys.sj3.marketscore.com	CubMail	jujube	2004-12-06 01:11:11
user2131	66.119.34.39	proxys.ia2.marketscore.com	CubMail	durian	2004-12-06 01:16:47
user317	170.224.224.102	proxys.or3.marketscore.com	CubMail	passionfruit	2004-12-06 08:35:15
user2113	170.224.224.70	proxys.or2.marketscore.com	CubMail	jujube	2004-12-06 09:04:10
user2102	216.148.244.70	proxys.sj2.marketscore.com	CubMail	jujube	2004-12-06 09:18:59
user2113	170.224.244.70	proxys.or2.marketscore.com	CubMail	jujube	2004-12-06 09:19:18
user55	216.148.246.134	proxys.sj4.marketscore.com	CubMail	passionfruit	2004-12-06 09:33.42

Figure 2: Sample MarketScore logins (users obfuscated).

We will improve the user-facing application to query personal login information. Any user tools that decrease support staff time are a boon.

We will further evaluate possible machine learning algorithms with more satisfactory error rates, and possibly incorporate these algorithms into the user-facing tool.

We will look at using real-time log-processing frameworks, such as SHARP, to collect information as it is available [14].

We will further research more widely-used standards for sharing logging messages, such as Internet2's ccBAY and Conostix's IPFC [18, 19].

Availability

This paper and related code can be found online at http://www.columbia.edu/acis/networks/advanced/gulp/ .

Author Information

Matt Selsky earned his BS in Computer Science from Columbia University. He has been working at Columbia University since 1999, most recently as an engineer in the UNIX Systems Group. He works on e-mail-related services and is currently pursuing an MS in Computer Science from Columbia University. Reach him electronically at selsky@columbia.edu.

Daniel Medina completed his BS and MS in Computer Science at Columbia University. Since 2002, he's worked in the Network Systems Group at Columbia University. He can be reached at medina@columbia.edu.

Bibliography

[1] *Kerberos: The Network Authentication Protocol*, http://mit.edu/kerberos/ , Accessed 7 December, 2004.

[2] "Columbia's Central UNIX Hosts," http://www.columbia.edu/acis/sy/cunix/ , Accessed 2 December, 2004.

[3] *Restricting Access: WIND*, http://www.columbia.edu/acis/webdev/wind.html , Accessed 2 December, 2004.

[4] *IMP Webmail Client*, http://www.horde.org/imp/ , Accessed 6 December, 2004.

[5] *wtmpx – utmpx and wtmpx database entry formats*, Sun Online Manual Pages, Accessed 2 December, 2004.

[6] *Apache HTTP Server Log Files*, http://httpd.apache.org/docs/logs.html , Accessed 6 December, 2004.

[7] *onall – Run a command on a group of hosts*, AcIS Online Manual Pages, Accessed 2 December, 2004.

[8] Kundakci, Vace, *'Free Love' and Secured Services*, EDUCAUSE Review, pp. 66-67, http://www.educause.edu/ir/library/pdf/ERM0266.pdf , Nov/Dec, 2002.

[9] *doxdesk.com: Parasite: MarketScore*, http://www.doxdesk.com/parasite/MarketScore.html , Accessed 6 December, 2004.

[10] *MySQL: The World's Most Popular Open Source Database*, http://www.mysql.com/ .

[11] *Whois Proxy*, http://grove.ufl.edu/bro/ , Accessed 6 December, 2004.

[12] Rekhter, Y. and T. Li, *RFC 1771: A Border Gateway Protocol 4 (BGP-4)*, March, 1995.

[13] Takada, T. and H. Koike, "Tudumi: Information Visualization System for Monitoring and Auditing Computer Logs," *Proceedings of the 6th International Conference on Information Visualization (IV '02)*, July, 2002.

[14] Bing, M. and C. Erickson, "Extending UNIX System Logging with SHARP," *Proceedings of the 14th Systems Administration Conference (LISA 2000)*, December, 2000.

[15] Sah, A., "A New Architecture for Managing Enterprise Log Data," *Proceedings of the 16th Systems Administration Conference (LISA '02)*, November, 2002.

[16] Takada, T. and H. Koike, "MieLog: A Highly Interactive Visual Log Browser Using Information Visualization and Statistical Analysis," *Proceedings of the 16th Systems Administration Conference (LISA '02)*, November, 2002.

[17] Finke, J., "Monitoring Usage of Workstations with a Relational Database," *Proceedings of the 8th Systems Administration Conference (LISA '94)*, September, 1994.

[18] Internet2, *MW-E2ED Diagnostic Backplane Pilot Effort (ccBay)*, http://middleware.internet2.edu/e2ed/public/pilot/pilothome.html , Accessed 29 July, 2005.

[19] Conostix S. A., *IPFC (Inter Protocol Flexible Control)*, http://www.conostix.com/ipfc/ , Accessed 25 July, 2005.

Appendix

XML Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:gulp="http://www.columbia.edu/xml/gulp"
 targetNamespace="http://www.columbia.edu/xml/gulp"
 elementFormDefault="qualified">

<xsd:annotation>
    <xsd:documentation xml:lang="en">
     [G]rand [U]nified [L]ogging [P]roject schema for AcIS
    </xsd:documentation>
</xsd:annotation>

<xsd:simpleType name="UNI">
 <xsd:restriction base="xsd:string">
   <xsd:pattern value="==>[ignored: w]<==+"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="timestamp">
 <xsd:restriction base="xsd:string">
   <xsd:pattern value="(\d{4}\d-\d{2}-\d{2}\ds\d{2}:\d{2}:\d{2})?" />
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ip_addr">
 <xsd:restriction base="xsd:string">
   <xsd:pattern value="[\d\.]+" />
 </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="bLog">
 <xsd:all>
   <xsd:element name="uni"          type="gulp:UNI"/>
   <xsd:element name="starttime"    type="gulp:timestamp"/>
   <xsd:element name="endtime"      type="gulp:timestamp"/>
   <xsd:element name="service"      type="xsd:string"/>
   <xsd:element name="server"       type="xsd:string"/>
   <xsd:element name="remote_addr"  type="gulp:ip_addr"/>
   <xsd:element name="remote_hname" type="xsd:string"/>
   <xsd:element name="note"         type="xsd:string"/>
 </xsd:all>
</xsd:complexType>

<xsd:element name="logs">
<xsd:complexType>
 <xsd:sequence>
  <xsd:element name="basicLog" type="gulp:bLog"
                               maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>
</xsd:element>

</xsd:schema>
```

Database Schema

```
create table basic_log (
        uni varchar(8) not null,
        starttime datetime,
        endtime datetime,
        -- well-defined service name
        service varchar(25) not null,
        -- server node name
        server varchar(20) not null,
        remote_addr int unsigned not null,
        remote_hname varchar(75),
```

```
        note varchar(200),
        id int not null auto_increment,
        primary key(id)
);

create index basic_log_uni on basic_log(uni);
create index basic_log_remote_hname on basic_log(remote_hname);
create index basic_log_remote_addr on basic_log(remote_addr);
create index basic_log_service on basic_log(service);
```

Toward an Automated Vulnerability Comparison of Open Source IMAP Servers

Chaos Golubitsky – Carnegie Mellon University

ABSTRACT

The attack surface concept provides a means of discussing the susceptibility of software to as-yet-unknown attacks. A system's attack surface encompasses the methods the system makes available to an attacker, and the system resources which can be used to further an attack. A measurement of the size of the attack surface could be used to compare the security of multiple systems which perform the same function.

The Internet Message Access Protocol (IMAP) has been in existence for over a decade. Relative to HTTP or SMTP, IMAP is a niche protocol, but IMAP servers are widely deployed nonetheless. There are three popular open source UNIX IMAP servers – UW-IMAP, Cyrus, and Courier-IMAP – and there has not been a formal security comparison between them.

In this paper, I use attack surfaces to compare the relative security risks posed by these three products. I undertake this evaluation in service of two complementary goals: to provide an honest examination of the security postures and risks of the three servers, and to advance the study of attack surfaces by performing an automated attack surface measurement using a methodology based on counting entry and exit points in the code.

Introduction

System administrators frequently confront the problem of selecting a software package to perform a desired function. Many considerations affect this decision, including functionality, ease of installation, software support, interoperability, performance, and hardware and software dependencies. However, the sensible administrator will place significant weight on choosing a software package with a reasonable posture towards security.

When selecting a package and installing it initially, the administrator can ensure relatively easily that the new configuration is not susceptible to any known vulnerabilities. However, some software is more prone to future attacks than others. Sysadmins could benefit from an easy-to-implement metric which provides an intelligent assessment of which software packages are likely to be found vulnerable in the future, not just of which ones have had problems in the past.

Measurement of the attack surface of a software package has been proposed as such a metric. The attack surface is the set of opportunities which the software package provides for the outside world to access the software itself. It is measured by enumerating this set and classifying its elements on the basis of risk of future attack. Researchers have further proposed automating attack surface measurement by enumerating and classifying the entry and exit points of the software package. In this paper, I describe steps taken towards performing an automated relative vulnerability assessment of open source UNIX IMAP servers based on measurement of attack surfaces.

Contributions and Roadmap

The paper makes two major contributions. First, I undertake an in-depth discussion of the relative security postures of the three major open source IMAP servers in use today. Second, I perform a partial measurement of attack surfaces based on the entry/exit point methodology, and compare the results to my initial impressions of the servers studied.

In the remainder of this section, I introduce attack surfaces and IMAP servers. I discuss the prior work done on attack surface measurement and the rationale for selecting IMAP servers as the target of my analysis. I discuss IMAP servers at the design level by first exploring the high-level interactions of the servers with their environments, then introducing the three target servers with a focus on the design philosophies and implementation peculiarities of each. In the next section, I introduce the entry/exit point method of attack surface measurement, and discuss the methodology used in performing a partial entry/exit analysis of the IMAP servers, including roadblocks to performance of a fully automated analysis. Finally, I compare the results of the analyses performed, and provide conclusions and future directions for attack surface research.

Attack Surfaces

Theory of Attack Surfaces

The attack surface can be used to create a comparative software vulnerability metric for packages which perform similar functions. Such a metric assesses a system at the design level by observing the prerequisites for a system to be attacked.

An attack on a system necessarily involves an attacker who gains access to some resource to which

he is not entitled. There are some prerequisites without which an attack cannot occur. First, the attacker must be able to take some action which affects the system. If he cannot alter a system's state at all, he cannot increase his access to that system. Second, in order for an attack to be non-trivial, the system must contain some resource which is accessible after the attack, but not before. This resource may itself be the target of the attack, or it may be an enabler which allows the attacker to reach his eventual target, or to get closer to it. For example, in the case in which an unauthenticated attacker triggers a buffer overflow in a port-listening daemon to obtain a root shell on the server, the action is the daemon, which runs code based on connections received from anyone, and the resource is the daemon's root privilege.

The attack surface of a system is simply the set of all actions made available by the system and all resources accessible to the system. This set consists of three types of items: (1) methods, which are executable code potentially runnable by an attacker, (2) channels, which are IPC mechanisms potentially usable by an attacker, and (3) data items, which are sources of persistent data (such as files) potentially readable or writable by an attacker.

However, enumerating system actions and resources is not very useful in itself, because it tells us nothing about the specific risks posed by the actions and the specific opportunities afforded by the resources. A port-listening daemon which requires connections to come from a specific IP address and which asks clients to cryptographically authenticate themselves before communicating further may pose less risk than does a daemon which will allow any client to talk to it. We need some way to discuss the relative risks posed by different methods or resources.

Prior Work on Attack Surface Measurement

The idea of attack surfaces was introduced by Howard, and was used to measure the relative attack surfaces of versions of Windows [11]. Manadhata and Wing extended the description of attack surfaces in 2004, and discussed ways of measuring attack surfaces in the context of Linux versions [14]. Their approach to the problem that not all system resources are created equal was to use the Common Vulnerabilities and Exposures (CVE) database [1] to identify several types of objects which were most often involved in attacks on Linux. Typed objects include things like "http daemon running as root" or "file writable by group users." They then enumerated those types within instances of each of several Linux distributions to draw a comparison.

Several improvements on their analysis are envisioned here. First, manual analysis is tedious and error-prone, so a means of automating attack surface measurement is desired. Second, it is not possible to use this method to compare attack surfaces between two products – if one has more daemons running as root, but the other has more world-writable files, we cannot say anything further about which poses the greater risk. Third, identifying important resources based on previously discovered vulnerabilities assumes both that the set of known vulnerabilities is large enough to be representative[1] and that future attacks will exploit the same types of resources and actions as previous attacks. In order to focus on features which could be attacked in the future, we need to define types in a way which more accurately assesses the risk introduced by each. The entry/exit point analysis discussed in the third section of this paper attempts to address these issues.

IMAP Servers

The Internet Message Access Protocol provides e-mail access to authenticated remote users. Once a Mail Transfer Agent receives a message for a local recipient, it must place that message into a data store from which the recipient can access it at his leisure. In many cases, the MTA places the message directly into a file, using a mail storage format such as Maildir or Berkeley mbox. The recipient then logs onto a UNIX system which has access to the mail file, and uses a UNIX-based mail client (MUA) to read the message.

IMAP provides a mechanism for the mail store to be accessed over a network. The MTA hands a message off to the IMAP server, either by storing it in a file format the server can read, or by communicating with the IMAP server using a mail transmission protocol such as SMTP or LMTP (Local Mail Transfer Protocol). At a later time, the recipient may connect to the IMAP server over a network, authenticate, and perform actions such as reading new messages, deleting messages, or selecting messages from his mailbox based on certain criteria [15].

I chose to investigate IMAP for several reasons. First, there are three freely-available open source IMAP servers which share the bulk of the market – UW-IMAP, Cyrus, and Courier-IMAP. Each server has been in development for several years, so the codebases are fairly mature. Each codebase contains 100,000-250,000 lines of code. These codebases are large enough that it is not practical to perform an analysis by reading every line of the code, but small enough that it is possible to get a design-level sense of the actions the code takes. Second, since IMAP is a niche protocol relative to, for instance, SMTP or HTTP, there has not been a formal security analysis of IMAP servers in the past. Third, the IMAP protocol is stateful, and, in particular, provides for both authenticated and unauthenticated states. In theory, attack surfaces are well suited for analysing stateful protocols, because they take into account what privileges are prerequisites for attacking a certain portion of a system. For all these reasons, studying the attack surfaces of IMAP

[1] This is certainly not the case for IMAP servers, for which only about 30 distinct server vulnerabilities are reported in the CVE.

servers is both feasible and worthwhile, in that it provides useful information about server security.

Observation of IMAP Server Software

To assess the effectiveness of the attack surface analysis, I needed to gain a subjective impression of the relative security postures of the three IMAP server packages. In order both to increase my knowledge of IMAP server operation, and to build a consistent baseline from which to compare the codebases, I installed each server package in a constrained environment. To the extent possible, I applied the same minimal configuration to each server.

Observation Methodology

I installed each package using the jail() system call under FreeBSD 5.2. The jail() call is an extension of chroot() which restricts a called program and its children to operating within a subdirectory of the filesystem. All dependencies of the jailed program, including libraries, configuration files, devices, and log files and sockets, must exist within that subdirectory. In addition, jail() binds all network activity of the jailed process and its children to one given IP address [12].

Using jail(), I was able to create a miniature virtual server for each IMAP package. Each virtual server ran only three items: a syslog daemon providing debug-level system logging for all activity, a listener of some sort to receive e-mail via LMTP connections, and the IMAP daemon itself. In addition, I created a virtual server containing the Postfix MTA, whose purpose was to forward mail via LMTP to each of the three IMAP servers.

For each IMAP server, I attempted to create a minimal default configuration in which the server listened only on port 143 for both encrypted and unencrypted connections.[2] Each server provided access to one user account, which could be authenticated via a CRAM-MD5 database.

I determined the minimal set of files which each server needed in order to function by trial and error. I modified the jail's startup routine to start all programs as children of ktrace(), and used that in combination with syslog error output to iteratively determine what additional files were needed. Needless to say, this procedure does not give an entirely comprehensive view of all the files a given program might use under any circumstances. However, it does enable some subjective assessment of the behavior of each package, including: installed size and dependencies, behavior during compilation and configuration, network and system behavior during operation, and any notable extra features the software provided, be these user-visible options, extra daemons and port listeners, or API methods.

[2]Technically, the port listens only for unencrypted connections. However, it is possible for a client to connect and immediately issue the STARTTLS command, thereby encrypting the remainder of the session.

In the remainder of this section, I will introduce each of the three IMAP servers in turn, discussing the project overview of the package, the code layout and high-level design, my experience in installing and configuring the package, and my subjective impression of the security of the package.

High-Level Interactions of IMAP Servers

Some high-level elements are common to all IMAP servers, as a result of the services they provide. In particular, there are several ways in which an IMAP server must interact with its environment in order to do its job. These interactions correspond to types of attack surface elements which all IMAP servers must have, so it is worthwhile to outline them in a general sense before proceeding to the specifics of each server package. Figure 1 depicts the interactions described in the remainder of this section.

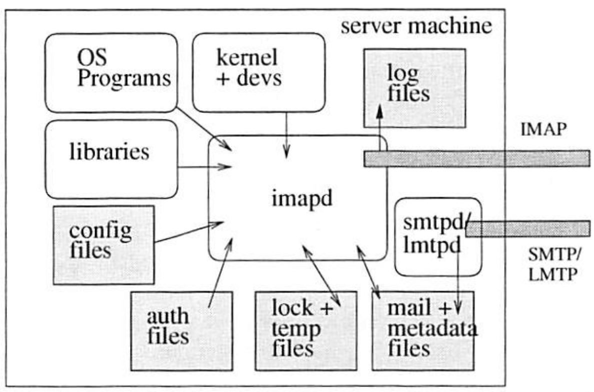


Figure 1: High-level IMAP server interactions and dependencies. (Executables are white, files are grey. Imapd may provide lmtpd service internally).

Remote API

First and foremost, the IMAP server provides a network API (application program interface) to a remote user. The server listens on a port – commonly port 143 for unencrypted or encrypted connections, or port 993 for encrypted connections only – and responds to user connection requests.

The IMAP API is the set of commands which the IMAP server considers to be legal for the remote client to send. It consists of 30-50 commands, the majority of which are specified by the IMAP RFC. The user sends these commands to the server using a fixed format and a command-specific number of arguments, and each command has an RFC-defined effect. The command set includes actions such as CAPABILITY (list the features offered by this IMAP server), STARTTLS (negotiate encryption for this connection), AUTHENTICATE (begin some form of negotiation to move from an unauthenticated state to an authenticated state), and SELECT (choose a mailbox for further operations).

Most of the commands deal directly with processing of e-mail, and thus should be accessible only

to authenticated users. However, a set of authentication commands must be available to anyone who connects to the server, so that authentication can take place. In addition, some servers offer support for folders available to anonymous users, and thus for an anonymous login mechanism.[3]

Access to User E-Mail Data

In order to serve e-mail, the IMAP server must have a way of receiving that mail from the MTA. The e-mail reception mechanism can take one of two forms: either the MTA deposits the mail in a format the IMAP server can read, or the IMAP server listens over a network or named socket for connections from the MTA. In the former case, the MTA and the IMAP server must share disk access. In the latter case, the IMAP server must run another listening daemon which speaks a mail transfer protocol such as SMTP or LMTP.

Once the mail has been received, the IMAP server must be able to read and modify the messages on disk, and must maintain e-mail metadata in order to keep track of mailbox state required by the IMAP protocol. Typically, metadata is stored as separate files associated with each user or each mailbox, or as header data within mail files.

Operating System Dependencies

The IMAP server depends on its resident operating system (a version of UNIX in the cases of my three reference servers) to provide it with several necessary functions. First, there is a standard set of utilities and system libraries – such as ls and libc – which are needed by almost any running program.

Second, the IMAP server needs a way to listen for incoming connections over the network. Some servers implement their own networking code, while some take advantage of OS-provided daemons such as inetd to launch the IMAP server in response to connection requests.

Third, each IMAP server uses some third-party libraries to provide various functions. All three servers make use of OpenSSL to encrypt the communication channel with the remote user. In addition, some of the servers use libraries such as BerkeleyDB to process authentication or metadata files in database formats.

Access to Non-Mail Files

Lastly, each server makes use of a number of files which are not directly accessible to the remote user or related to e-mail processing, but are used by the server backend. These files are of four primary types: (1) configuration files to be read at server startup or during operation, (2) log files or a logging subsystem to be written with error and/or debugging output, (3) authentication files or an authentication subsystem to be consulted when a remote user attempts to prove his identity, and (4) lock files and temporary files to be written in order to store non-permanent data or to control access to shared elements.

[3]This mechanism typically allows for login with the username anonymous and any valid text string as a password, and should be familiar to users of FTP.

UW-IMAP Server

Overview

UW-IMAP [9] is written and maintained at the University of Washington by Mark Crispin, the author of the original IMAP RFC. The purpose of this package is to provide a simple and flexible drop-in IMAP server for multi-user systems. The package uses the assumption that IMAP will be one of many login methods through which remote users can access the system. In particular, the functional differences between IMAP access and a shell access method such as SSH should be only that IMAP access is optimized for mail reading. Restricting IMAP access beyond the access afforded to a shell user is not a design goal.

The UW-IMAP server has been under active development since 1988, though the entire codebase has been rewritten several times since then. The current code is considered to go back only as far as the 2000 imap-2000 release. Looking further back, I find a code overlap of approximately 20% between imap-2004c1 (the most recent version as of this writing) and the 1996 imap-4 release, and no overlap between imap-2004c1 and any release prior to imap-4.

The current codebase contains 135,000 lines of code and 40,000 lines of other files. Of this code, the IMAP server itself comprises only 4,000 lines, while the remainder of the code consists of an internal (compiled-in) library called c-client. This library is also the backend for the Pine e-mail client.

Compiling imapd provides a single binary with a single purpose. An external program such as inetd must be used to listen on the appropriate IMAP ports. When a connection is made, an imapd process is spawned, handles that single connection, then terminates. Since UW imapd's place in the system is simple, the amount of code needed for its implementation is reduced. The tradeoff is increased dependencies on other programs to perform core functions, most notably mail delivery and port listening. The imapd program also requires no configuration file – configuration options are to be selected at compile time.

One more notable feature of UW-IMAP is that it is agnostic about mailbox formats. By default, the UNIX UW installation is compiled with support for mbox, mbx, mx, mh, tenex, mtx, mmdf, and phile mailbox types. This support is provided by means of mailbox drivers. Internal logic is used to guess the type of a mailbox, and then execution is passed off to the appropriate driver.

Impressions

The UW codebase does not have a good security history, and the design does not inspire confidence about

its future. My survey of IMAP vulnerabilities found in the CVE database (through December 2004) found twelve relevant vulnerabilities in the UW server, compared to seven in Cyrus and four in Courier. UW-IMAP's primary benefit is that it is the smallest and simplest of the three servers, both in terms of code size and major functions provided, and in that it provides a smaller set of IMAP API methods than the other servers. (The small API set may be in part due to the fact that the UW author wrote the IMAP RFC, which defines the minimal allowable set of API functions.)

However, the drawbacks are many, and seem to go down to the design philosophy of the package. The code is not at all modular. An example of this is given by the imapd program's main() routine. The routine contains code to select a mailbox sorting function for the SORT API method. It also contains three separate code locations at which the anonymous user is configured (a pre-authentication check at the beginning of the connection, an outcome of the LOGIN API method, and an outcome of the AUTHENTICATE API method), causing three distinct variables to be set each time. In addition, since the codebase is so non-modular, and since most of the functionality is provided by a c-client library which is also the backend for the mail client Pine, it is possible that functionality may be compiled in to the UW server which is really only necessary or desirable for client operation.

The design decision that IMAP access should be only one method into a user-accessible system effectively prevents administrators from building a closed box IMAP server. (UW can be compiled in a mode called closedBox, but the differences from the standard configuration are limited, and this is not an officially supported configuration.) The assumption that the server will not be a closed box shows up in several places. For example, one of the methods by which the Pine client attempts to contact an IMAP server uses rsh or ssh to run a server instance locally. In order to support this, the imapd server offers a "pre-authenticated" mode, in which a server started under a given UID is assumed to be authenticated as the associated IMAP user.

Despite UW-IMAP's history of buffer overflows, instances of string functions which do not perform length-checking (such as sprintf) are still plentiful within the code. The codebase uses custom makefiles, rather than GNU configure or some other standard solution, to maintain the cross-platform build process. This choice is not necessarily related to security, but it causes code compilation to be less uniform than might otherwise be the case. Along the lines of the closedBox option, the package officially disallows use of a configuration file, but the code specifies the name of a configuration file which will be read if it exists. According to the documentation, the results of using this file to configure the software are unpredictable.

Cyrus Server

Overview

Cyrus is written and maintained by Project Cyrus at Carnegie Mellon University [4]. Its purpose is to provide fast and scalable IMAP service. In order to support this goal effectively, the Cyrus server is, by design, truly a closed box. Cyrus permissions are defined internally and do not map to the UNIX permissions of the host operating system, and Cyrus uses its own custom mailbox format which is not guaranteed to be parsable by non-Cyrus tools.

Cyrus has been under active development since 1994. In approximately 1999, the codebase was split into cyrus-imap, the IMAP server, and cyrus-sasl, a set of flexible authentication libraries and associated utilities for use with IMAP or other Cyrus programs.

The Cyrus codebase contains 210,000 lines of code and 475,000 lines of other files. It is therefore the bulkiest of the three codebases, but is also relatively well documented. Of the included code, 75,000 lines come from the SASL codebase (the wrapper authentication libraries themselves, the optional authentication daemon, plugins for common authentication mechanisms, and utilities for checking and changing passwords), while the remainder is the IMAP codebase. The IMAP side of the code provides a number of auxiliary tools and functions, but the code specific to imapd itself is nearly 70,000 lines.

Compiling IMAP and SASL produces a large number of binaries and libraries, many of them optional. In order to function at all, the system requires: the front-end daemon master to listen for incoming network connections and pass them off to the appropriate servers, the imapd binary to handle IMAP connections, the lmtpd binary to receive incoming mail from the MTA and store it in the Cyrus mail format, and the ctl_cyrusdb binary to maintain Cyrus's extensive collection of mail metadata. The system also requires the primary SASL library libsasl2, as well as some helper libraries which implement particular authentication mechanisms. Cyrus is configured using the files cyrus.conf, which specifies the listeners and periodic processes spawned by master, and imapd.conf, which configures IMAP-specific options.

In order to support fast and scalable service, all Cyrus code runs under a single UNIX user account. It is possible to use the UNIX passwd file for authentication of IMAP users, but doing so requires the use of a special daemon, saslauthd, which runs as root and with which the SASL authentication mechanisms communicate via a named socket.

The other major Cyrus design decision which contributes to scalability is the custom mailbox format. Individual e-mail messages are stored in a readable format in flat files, one file per message contained in one directory per mail folder. (This directory

structure is similar to that of Maildir.) Mail metadata is stored in binary database files using a Cyrus-specific format, and thus cannot necessarily be read by non-Cyrus tools.

These two design choices bring with them the requirement that all access to or support of user mail be performed using Cyrus tools, and typically over the network. As a result, mail administration commands are provided via the remote IMAP interface, and a special subsystem, called sieve,[4] is provided for end user mail filtering tasks which other systems handle by allowing users to install procmail filters. When installed, sieve adds extra code, as well as some mechanism for end users to edit their filters – the most common is an extra port listener through which users can authenticate and make changes over the network.

Impressions

I found Cyrus to be the most impressive of the codebases in terms of layout. The code is modular and well documented, and can be easily compiled using GNU configure.

Since the IMAP server is designed to be run in a black box, any functionality not explicitly provided should be denied. Therefore, Project Cyrus takes breaches very seriously. From a security perspective, the design feature by which Cyrus runs all its code as the imapd user has both a major benefit and a serious drawback. The benefit is that no code (other than the optional saslauthd) runs as root, so it should never be possible to gain root access on the Cyrus box solely by compromising the Cyrus server. Therefore, if your primary concern is the prevention of root compromises on your network, the Cyrus methodology is a very good one. However, the downside is that any arbitrary code execution vulnerability constitutes a total breach of the IMAP system. If your primary concern is the safety of the mail system itself, you might prefer a system in which a buffer overflow in post-authentication code gave the authenticated user access only to his own mail, rather than to all mail and mail metadata on the system.

Cyrus provides a large number of features internally, some of which (such as mail administration and filtering) are required by the design of the system, but some of which seem like evidence of feature creep. For instance, there is a notification daemon to provide flexible user notification of new e-mail. Additionally, Cyrus, like UW, has a built-in NNTP client, and provides anonymous user access for the purpose of using the IMAP server as an NNTP aggregator.

Obviously, the in-band mail administration is itself risky, since it opens up more code to the IMAP interface. This risk is somewhat mitigated by the clean and modular design of the IMAP-visible code, but cannot be entirely removed. The Cyrus system is also busier than the others – the master process spawns periodic tasks related to cleaning and optimizing the database, while UW and Courier's daemons perform no work unless a connection is being served.

Courier-IMAP Server

Overview

Courier-IMAP is the IMAP server component of the Courier MTA [2]. The IMAP server is packaged with the MTA, but is also available and configurable as a stand-alone server. Courier is designed to be fast and scalable while being interoperable with standard UNIX permission schemes and mailbox formats. It attains this balance by using the Maildir format, which is a one-message-per-file format similar to Cyrus, but is supported by many other applications, including several MTAs.

Courier-IMAP has been in development since 1998, and has been a continuous codebase since that time. Like Cyrus, Courier is divided into an IMAP portion of the codebase and an authentication library, called courier-authlib. However, the imap/authlib split was recent in this case, dating back only as far as the 2004 release of courier-imap-4.0.0.

The current codebase contains 135,000 lines of code and 550,000 lines of other files, so it is relatively lean and also well documented. Approximately 30,000 lines of code are represented by the authentication library, and the rest by IMAP. The IMAP codebase includes various compiled-in helper libraries, such as Unicode handlers, support for various mail-relevant RFCs, and support for the Maildir format. The daemon code itself is divided into: a network listener which provides the functionality of inetd and tcpwrappers in a Courier-specific fashion (couriertcpd), a binary designed for handling unauthenticated IMAP connections (imaplogin), and a binary which should only ever be run by authenticated IMAP connections (imapd).

In addition, Courier-IMAP requires its own syslog frontend (courierlogger), several libraries related to authlib, and a mandatory authentication daemon (authdaemond), which listens on a socket. (By contrast to Cyrus, Courier requires authdaemond even if UNIX passwords are not used for authentication.)

The Courier server has as a design goal protecting the system from its own end users (legitimate or otherwise). Therefore, it has a number of configuration settings related to limiting the system resources available to IMAP users.

Impressions

Courier-IMAP has many positive security-relevant features, and, at an overview level, seems very well designed. The code is relatively minimal in many important ways – no NNTP client or anonymous access is provided. Privilege separation is cleanly implemented at a conceptual level. For instance, Courier requires a separate configuration file to run imapd on its non-SSL

[4] Sieve is an open standard mail filtering language, defined by RFC 3028.

port and on its SSL port, so it is very easy to configure only the secure port with no risk of accidentally enabling insecure access. Also, the login design, in which the pre-authenticated IMAP connection is handled by an entirely separate binary from the one which provides logged-in functions, is very clean.

Upon closer inspection, however, many things in the code disappoint. To increase efficiency, the imaplogin process does not always terminate on unsuccessful connections. The way in which this is implemented potentially opens opportunities for the connection to be left in a partially logged-in state. A bug in this portion of the code might allow an attacker to circumvent the privilege separation. In addition, the code layout itself is hard to follow and inconsistently modular.

Compilation and configuration were difficult and seemingly broken – configure is run once for each subdirectory of the compile directory (rather than maintaining a cache of, for instance, the location of gcc, or the size of an integer), leading to slow and repetitive compilation. More seriously, the compile-time configuration takes some values from outside the configuration directory in an undocumented manner. Running make clean in the Courier directory removes files which are required for compilation, so it is necessary to restore from the original tarball in order to redo a compilation. None of these items is relevant to the running system, per se, and each may be attributable to the recent imap/authlib code split, but they are inconvenient and do not inspire confidence in the package design.

In addition, there are some features which do not seem like good ideas. The cyrus-sasl implementation which allows non-daemon authentication mechanisms seems preferable to the courier-authlib implementation which always requires a daemon, thereby mandating that the system be open to attacks against the daemon itself. The server also provides some strange user features. Most notable among these are: INBOX.Outbox, a special user-visible mailbox into which users can place messages for immediate transmission via SMTP, and loginexec, a file within a user's top-level Maildir which is always executed and removed, regardless of ownership, if it exists.

Summary of Software Observation Results

It is certainly the case that manual software observation gives an incomplete impression of software security. The overall code layout, modularity, and privilege separation are obscured from the time-constrained examiner, who sees only the set of files installed and the modularity of the main() loop.

The most visible features are this high-level view of the code, the ease of installation and compilation, and the presence and quality of documentation (particularly the subset used to assist the initial installation). It is easy to find the set of mandatory requirements and dependencies of each package in the chosen configuration, but impossible to tell what additional requirements other configurations might impose.

In addition, any extra features which are offered to the user or provided in the software's back end are very visible. If such features appear to be security risks, their effect on the examiner's opinion of the code may be disproportionate to their effect on the actual code security. In this particular case, my views on the codebases were strongly affected by the defensive attitude towards security concerns taken by the UW-IMAP documentation, and by the presence of strange features in the Courier-IMAP codebase.

Based on manual software observation alone, Cyrus appeared likely to be the most secure and UW likely to be the least secure of the three packages.

Entry/Exit Point Analysis

In this section of the paper, I return to the formal definition of attack surfaces, and describe steps taken towards a methodical measurement of the attack surfaces of my testbed IMAP servers. I report on the partial results obtained from my analysis, and discuss how the results compare to those derived by manual examination. This comparison is useful both because of what it says about IMAP servers per se, and because, in order to use entry/exit point analysis to measure attack surfaces, we need to ensure that the results obtained from that analysis are meaningful.

In addition, my analysis is also interesting in terms of the obstacles to automated measurement which I encountered. The problems encountered in this analysis, some technical and some related to the analysis methodology, need to be solved in order to meaningfully automate the measurement of the attack surface of a given codebase.

Entry/Exit Background

The entry/exit methodology endeavors to provide two major developments in the measurement of attack surfaces. The first is the ability to automate the discovery of attack surface elements. The second is the ability to numerically compare the attack surfaces of different codebases.

Automated Discovery of Attack Surface Elements

Entry/exit analysis utilizes the following simple insight into attack surface measurement: In order to launch an attack on a system, an attacker must either transmit data into the system or receive data from the system [13].

Suppose we want to look at a codebase and, at the code level, find all the places which might be part of the attack surface. Any place in the code which is part of the attack surface must contain a call which receives data from or transmits data to the outside of the system. Therefore, if we find all such points, then we have found all positions on the attack surface, and we need only verify that each such point is actually an

attack surface element, and classify each one into an attack class (a set of attack surface elements which pose the same level of risk to the system).

This opens the door for automated attack surface measurement, since it should be possible to find entry and exit points in a manner which is at least somewhat automated.

Numerical Comparison of Attack Surfaces

One major difficulty with the attack surfaces discovered by a more ad hoc analysis is that they cannot be compared to one another, because there is no clear way to numerically order the features measured. However, if we use entry and exit points to find positions within the code at which attack surface elements occur, our task becomes easier.

Suppose that, for each position in the code, we can determine what access rights are needed to run that code, and what privileges the running code has. Then, we have a one-dimensional set of access rights which are directly comparable, and a similar set of privileges. Moreover, the relative ordering we should use on these elements is clear: it is more difficult to gain administrator access than it is to gain user access, and it is more difficult to gain user access than it is to gain unauthenticated access. Similarly, root is more powerful than an imapd system user, which is more powerful than a normal user, which is more powerful than a service account.

We can then assign to privilege and access levels numerical values which are consistent with this ordering. Given these, we compute the attackability of a given attack surface item by observing that higher privilege items are more desirable targets (more attackable), and that higher access-right items are more difficult to target (less attackable). If we have defined functions

$$p : \{\text{privileges}\} \rightarrow \Re$$

and

$$ac : \{\text{access\_rights}\} \rightarrow \Re\ ,$$

and if a given item has privilege Q and access rights B, then we can define the attackability of that item as

$$\text{attackability} = \frac{p(Q)}{ac(B)}.$$ [5]

Therefore, if we have two different items with comparable privilege and access levels, we can numerically compare them in terms of attackability, and we can sum over all the attackabilities in a given codebase's attack surface, and compare the sum to that of another codebase.

However, we cannot actually measure attackability as a one-dimensional quantity. This is because the attack surface, as noted in the introductory sections, consists of three different types of elements – methods, data, and channels – and there is no straightforward way to correlate elements of these three types. Therefore, entry/exit analysis should ideally yield a three-dimensional vector describing the system attackability, data attackability, and channel attackability of a codebase.

Methodology for Entry/Exit Point Measurement

In this section, I discuss both the methodology I used for the successful measurement of method entry and exit points in the IMAP codebases, and my efforts to measure data and channel entry and exit points.

Measuring Data and Channels

The insight in measurement of data and channels is that, because of the way UNIX is designed, a process cannot read or write to a channel or persistent data object without making either a system call or a call to an external library which makes that system call itself.

Of course, we do not know the identities of all those external system or library calls. However, they are easy to find: a binary examination tool such as nm [8] will find the list of symbols which are referenced by a piece of compiled code. Then, a source code examination tool such as ctags [6] will give us a list of symbols actually defined within the code. Any symbols which show up in the nm output but not in the ctags output must be externally defined.

Once discovered, these external methods must be examined manually to determine whether they access data or channels, and what manner of access they perform (read, write, create, delete). However, in my experience, there were only about 150-200 unique external methods per codebase. Even better, these methods are provided by external libraries or by the operating system, rather than by the examined codebases. Therefore we can expect them to behave identically across codebases. Once the behavior of fopen() has been classified on a given operating system for one codebase, that classification can be used without modification for any other codebase running on that operating system. I was able to classify the calls made by my sample codebases, and to obtain a list of internal methods which made calls to external data and/or channels.

From there, the task became more difficult. The UW-IMAP codebase contained 896 instances in which an internal function made an external call involving data or channel access. That number might correspond to even more actual references. (For instance, a modular codebase might define an internal function whose job was to open a file and read from it. That function might be referenced from multiple different places in the code, each time with a different filename.) A very good code analyser might be able to find internal object names, leaving us with statements of the form "routine X read from the file whose name is stored in the variable foo at line 796." However, I did not have access to a code analyser of that quality, and, even so, this tells us nothing about which entities outside of the given codebase require permission to read or write the file, nor about whether the codebase's installation

[5] This definition is arbitrary, and is chosen because it is a simple function which varies directly with privilege and inversely with access rights.

procedure faithfully adheres to the strictest permissions possible rather than negligently substituting more relaxed permissions than needed.

We can instead work from the other end, and use software observation in a jailed environment to obtain the full list of files that each codebase requires in order to operate. However, this tells us only about the files required by the specific configuration used in the example jail, rather than the full set of files which might be required or used by the software.

Thus, using entry and exit points to count data and channel attack surface elements is not yet a solved problem.

Measuring Methods

This problem is more tractable. We would like to measure the set of all internal code which is runnable by an attacker. Provisionally, we measure code at the function level. First, we look at code which the attacker can call directly. Each of the three IMAP daemons uses a function with a name like main() to receive all input from the remote user, meaning that each daemon provides one reachable function. This is not very interesting.

Therefore, we can extend our analysis by looking at functions which are themselves called by those directly reachable functions, and are thus indirectly reachable by the attacker. We obtain these function names using a program flow analysis tool. In the case of IMAP analysis, I used a tool called cflow [7]. Display 1 shows some sample cflow output from the UW-IMAP case.

This gave several hundred lines of output per codebase, the name of every function reachable from the main() function. In order to count each method in the attack surface, I needed to find out what access rights it had, and with what privilege it ran. Some manual preparation was needed in order to obtain that information from the codebase.

On a UNIX system, the privilege with which the code runs is the privilege with which it started, unless a system call has occurred which changes privilege, such as setuid(). If a program starts running as root and later drops privilege, then any functions called before the setuid() call are called as root, and any functions called after setuid() are called as an unprivileged user.

It is slightly more difficult to determine the access needed to reach a certain element of the code, because it is necessary to find each code location at which authentication is performed. For instance, a call which compared the password provided by the user to one retrieved from a local file would constitute a change of authentication state – before that comparison, a user is unauthenticated, but after the comparison is successful, he is authenticated.

However the access is determined, once we know where the access and privilege changes occur, we can edit the directly reachable routine, creating a copy for each access and privilege pair which contains only those calls reachable from that level. Here is an example from my edit of the Cyrus codebase. Display 2 shows the original code fragment, from the part of the main loop which handles the AUTHENTICATE command.

In Cyrus, the variable imapd_userid is always set upon successful authentication, and at no other time. I edited the code fragment shown in Display 3 with that in mind, retaining only the code accessible to an unauthenticated user.

Note that I needed to verify that it is possible for authenticate_command() to return to the main loop upon failure. If authenticate_command() instead killed the process, or jumped to another segment of code, then I would have considered the call to snmp_increment() to also be unreachable by an unauthenticated user, and would have deleted it as well.

Once the edited copy had been made, I could then rerun cflow and obtain a subset of the original code elements, the subset reachable by the unauthenticated user. I then repeated this process for the Cyrus authenticated user, for the anonymous user, and for the administrative user, and for all combinations of access and privilege in each of the UW-IMAP and Courier codebases.

Obstacles to Entry/Exit Point Measurement

In the process of counting reachable methods, I encountered a number of obstacles, some technological, some integral to the enumeration process. I mention both types of difficulties, since technological problems might be solved by better tools, but, if those tools do not exist, the process of writing them offsets the time gains of automation.

Determining Whether Data Has Been Transmitted

In principle, we are not concerned with all functions reachable from the direct entry point, but only with functions through which the attacker can transmit the data required by his attack. How do we measure the subset of functions which actually allow the attacker to transmit data?

Clearly, if an attacker can input arbitrary data which is immediately read into an internal buffer belonging to the function, he has transmitted data into

```
main {src/imapd/imapd.c 253}
  strrchr {}
  mail_parameters {src/c-client/mail.c 297}
    fatal {}
    env_parameters {src/osdep/unix/env_unix.c 150}
      fs_give {src/osdep/unix/fs_unix.c 57}
       ... mail_parameters ... {3}
       free {}
```

Display 1: Partial cflow output from UW-IMAP codebase.

the function. However, consider a more restrictive case, in which, for instance, the input must have a certain format, such as being a valid SSL key, but is otherwise chosen by the attacker. Then consider the case in which the attacker can only insert a single integer value, such as a return code. Consider the case in which the attacker's return code is ignored by the calling function.

My conclusion was that the safest route is to count every called function, regardless of the syntax or content of the call. Display 4 shows an example which supports that conclusion. It is taken from the UW imap-2004a codebase[7] which was reported to be vulnerable in early 2005 [10] (all code irrelevant to the attack has been excised for clarity).

The problem here is caused by the logic used to set the variable u – the check done on md5try is backwards, so that, if md5try is 0 (if the attacker has tried to authenticate too many times using CRAM-MD5), the call automatically succeeds where it should automatically fail. The attacker does not need to send any particular exploit data in order to break into this system. All he needs to do is to attempt CRAM-MD5 authentication unsuccessfully three times, and then he will be authenticated on the fourth try.

In that light, it seems very reasonable to claim that causing a line of code to be run is sufficient, from an entry/exit point perspective, to transmit data into that code.

Undercounts and Overcounts

In order to automatically enumerate reachable methods, it is necessary to have an accurate method of detecting code execution paths. There are many tools which profess to do this, but, in practice, there are problems which lead to undercounts or overcounts.

First, in order to avoid undercounts, it is necessary to have a program which parses the code accurately. I used cflow for my final analysis because it employs gcc -E, and therefore works quite well.

[6] Cyrus-IMAP is copyright 1994-2000 Carnegie Mellon University. Some function and variable names have been modified at the request of CMU.

[7] UW-IMAP is copyright 1988-2004 University of Washington.

```
  ...
  if (imapd_userid) {
    protocol_printf(imapd_out,
      "%s BAD Already authenticated==>[ignored: r]<====>[ignored: n]<==", tag.s);
    continue;
  }
  authenticate_command(tag.s, arg1.s, haveinitresp ? arg2.s : NULL);
  snmp_increment(AUTHENTICATE_COUNT, 1);
} else if (!imapd_userid) goto nologin;
else if (!strcmp(command.s, "Append")) {
  if (c != ' ') goto argsmissing;
  ...
```

Display 2: Original Cyrus-IMAP authentication code fragment.[6]

```
  ...
  if (imapd_userid) {

  }
  authenticate_command(tag.s, arg1.s, haveinitresp ? arg2.s : NULL);
  snmp_increment(AUTHENTICATE_COUNT, 1);
} else if (!imapd_userid) goto nologin;
  ...
```

Display 3: Modified Cyrus-IMAP code: subset of code accessible to an unauthenticated user.

```
static int md5try = 3;

char *auth_md5_server (authresponse_t responder,int argc, char *argv[])
{
  char *ret = NIL;
  ...
  u = (md5try && strcmp (hash,hmac_md5 (chal,cl,p,pl))) ? NIL : user;
  ...
  if (u && authserver_login (u,authuser,argc,argv))
    ret = myusername ();
  else if (md5try) --md5try;
  ...
  if (!ret) sleep (3);     /* slow down possible cracker */
  return ret;
}
```

Display 4: Vulnerable CRAM-MD5 function in UW imap-2004a.

However, cflow ignores function arguments and program logic, which one might prefer to take into account.

For instance, in the following example, method_c is not actually reachable from method_a, but cflow would claim it was:

```
method_a() {
  method_b(DONT_USE_X);
}

method_b(int flag) {
  if (flag != DONT_USE_X) {
    method_c();
  }
}
```

Other tools which perform some of these functions include doxygen [5], which parses function arguments in a relatively usable fashion, but has a poor understanding of C syntax and does not deal with logic, and cqual [3], which is intended to handle both function arguments and program logic, but is difficult to work with and possibly buggy in practice.

A major source of undercounts in the IMAP case is caused by function pointers – variables which store the names of functions. At the point of execution, the function is referenced only by the name of the variable, so there is no reliable way to tell what actual function is being executed. All three IMAP codebases make use of function pointers, and no tool I found was capable of parsing them.

Classifying Multiply-Accessed Methods

Once the codebase has been divided by privilege and access levels, many functions are identified which are accessible by code running at different levels. This code may belong to internal or external helper routines, or it may be authentication-relevant code which has inadvertently been left too open. In order to classify the code, I needed to generate a strategy for handling such functions.

One strategy would be to count each function twice, once per privilege/access pair from which it was accessible. However, this seemed to overcount in the case of, for instance, an administrative user and a normal user who could both access normal user code.

Another approach would be to count only the worse of the two levels, i.e., if the function was accessible to either an unauthenticated user or to an authenticated user, count it as unauthenticated. However, this would probably provide an undercount, because an authenticated user might be able to reach more of the functionality of a routine than an unauthenticated user. If there are two attack paths related to a given function, both should be counted.

I compromised by creating new privilege and access levels to reflect multiply-accessible code. For instance, if root and user are privilege levels, then root+user would be the privilege level for code which could be reached at either of those levels. In the IMAP codebases, the appropriate total orderings of privilege and access levels were clear even with the additional levels.

Results of Entry/Exit Point Measurement

For each codebase, I identified the function responsible for receiving network connections from remote users, and broke down the codebase by privilege and access levels as previously described. For the UW and Cyrus codebases, this was a simple matter of finding the single input-handling function.

The Courier codebase was somewhat more complex. In that case, the program couriertcpd receives and processes an incoming network connection, does some checking, and then hands the connection off to imaplogin, which processes authentication routines. If imaplogin reports success, then an imapd process is spawned. Therefore, in order to find all the routines which could run at each privilege and each access level in the Courier case, I needed to analyse all three of these programs.

I also performed an analysis of Courier in which I omitted the couriertcpd code, which all runs as root and is accessible to unauthenticated users. Since the Courier and Cyrus codebases include network listeners in their codebases, while UW relies on inetd to provide network connectivity, I was interested in gauging the amount of code required to listen for network connections.

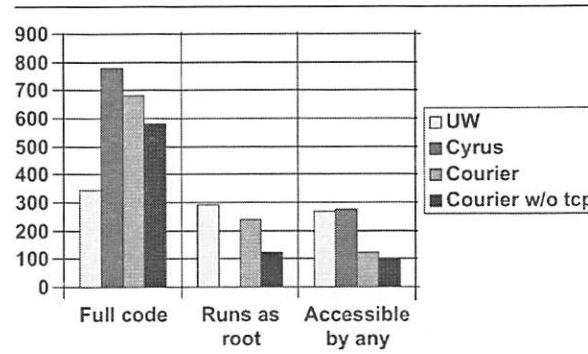


Figure 2: Number of reachable methods in each codebase: (a) full codebase, (b) code which runs as root, (c) code accessible at any access level.

Method Counts By Codebase

Figure 2 shows a representative subset of breakdowns of method counts among the codebases. The left graph shows the total number of methods reachable in each codebase. The middle graph shows the number of methods reachable by code which is running as root. (Note that Cyrus has no reachable code which runs as root.) The right graph shows the number of methods which are open to all access levels defined for the codebase. That is, in the UW case, this graphs functions which are reachable by an unauthenticated user, by an anonymous user, and by an authenticated user. In the Cyrus case, this graphs functions which the unauthenticated, anonymous, authenticated, and administrative users can all reach. In the Courier case, this graphs functions which the unauthenticated and authenticated users can both reach.

Total Orderings for Privilege and Access Rights

Once I had counted all reachable code methods, I needed to populate the total orderings for the privileges and access rights. All types in each ordering needed to be assigned numerical values, so that the attackability of each attack class, and thus of the entire system attack surface, could be computed.

Access Rights	Points
admin	8
auth	4
anon	1.5
auth + anon	1.45
admin + auth + anon	1.4
unauth	1
admin + unauth	0.95
auth + unauth	0.9
admin + anon + unauth	0.85
auth + anon + unauth	0.8
admin + auth + anon + unauth	0.75

Table 1: Attackability ordering for method access rights (higher value is harder to attack).

Privilege	Points
service	2
user	3
user + service	4
imapd	7
root	10
root + user	13
root + user + service	14

Table 2: Attackability ordering for method privileges (higher value is more valuable target).

Table 1 contains the ordering I derived for access rights, and Table 2 contains the ordering for privileges. Note that the ordering results naturally from known information about security of UNIX servers and the IMAP domain, but that the specific numerical values chosen are arbitrary.

System Attackability of IMAP Servers

Table 3 shows the system attackability of each IMAP server given these values and the reachable methods determined by the automated method. This table shows that, according to the attackability metric used here, Courier is the least vulnerable of the servers, while UW and Cyrus score similarly. The results also indicate that much of Courier's vulnerability is caused by its network listening code. This implies that Cyrus, which also contains network listening code, might be less vulnerable than UW were the functionality provided to be taken into account.

However, Figure 3 is worth noting. It demonstrates that the attackability metric is somewhat dependent on the exact numbers of points chosen for various privileges and access rights, even if the total ordering is held constant. By default, I assigned seven points to the special imapd user employed by Cyrus. However, a site which was most conscious of the need to avoid root compromise on their machines could assign imapd five points, close to the value for an unprivileged user, and would find that Cyrus significantly outperformed UW in attackability. A site with a different posture could note that any attack on the imapd user risked opening up all users' e-mail, and therefore was almost as bad as a root compromise. If this site assigned nine points to imapd, Cyrus would suddenly appear radically more attackable than UW.

Codebase	System Attackability
UW-IMAP	5044
Cyrus	5217
Courier	3122
Courier w/o tcp	1813

Table 3: System attackability of IMAP servers.

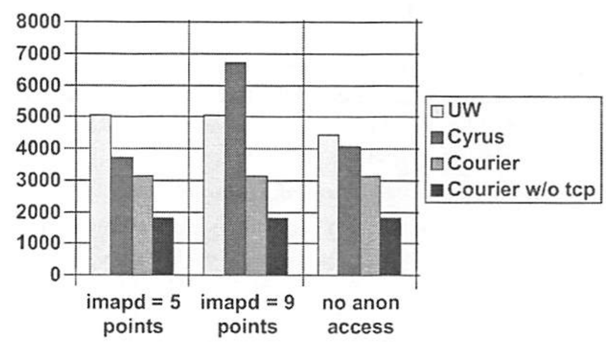


Figure 3: System Attackability of IMAP servers, with (a) imapd user worth 5 points, (b) imapd user worth 9 points, (c) imapd user worth 7 points and no anonymous access.

This tradeoff between protecting root and protecting user mail from other users is symptomatic of an obvious security consideration which arises when trying to decide whether to use Cyrus or a more conventionally-designed IMAP server. This example is included to demonstrate that the attackability metric does not protect us from having to make that judgment call.

In general, however, the attackability metric seems to agree reasonably well with observation. Despite the large size of the Cyrus codebase, its attackability is similar to that of UW-IMAP, indicating that Cyrus has good privilege separation while UW-IMAP does not. Indeed, the major effect of this metric as applied is to reward code with good privilege separation, which is a reasonable goal, though only one of many.

The couriertcpd example demonstrates that it is difficult to compare systems which perform different

functions, and that even between systems which are very similar in functionality, one system may provide many more of its own dependencies. In the process of measuring attackability, it is necessary to consider the external systems on which each codebase depends.

Conclusions

In this section, I discuss the feasibility of performing an automated entry/exit analysis as a means of comparing software security. I also discuss future work needed to determine whether entry/exit analysis of attack surfaces will be practical, and whether it will be useful. I close with a few words about the security of the specific IMAP servers I studied.

Feasibility of Entry/Exit Analysis

One important point is that it is difficult to ensure that any analysis is comprehensive. In the particular case of this analysis, I looked at only the code available via the network API, and did not examine the code added by periodic cleanup processes, LMTP, etc. I also did not look at the authentication daemons provided by Cyrus and Courier as part of the authentication facility. IMAP communicates with those via socket, so they would have surfaced had I done a channel analysis. It is still necessary to manually examine the code's functionality in some detail in order to find starting points for automated analysis.

In addition, entry/exit analysis is not trivial to perform. Approximately four to eight hours of labor was required to prepare each codebase for analysis by cflow, and that took into account significant familiarity with the code in question. The most time-consuming aspect is dividing the code based on privilege and access levels, but some time is also required to find the correct internal function definitions within a complex codebase, given that cflow and related tools cannot read Makefiles.

The division of the code is not entirely automatable work – some judgment calls are required in order to determine, for instance, at exactly what point authentication takes place. In theory, the Courier approach, in which a different physical piece of software handles execution at each privilege and access level, should make things easier. In practice, there are thorny issues even in that implementation, since authentication, setuid(), and exec() of the other daemon do not all take place simultaneously, nor even necessarily within the same function as one another.

In addition to all this, I did not attempt the data or channel enumerations, which would be more complex and difficult. In those cases, it might be complicated to even determine the total ordering among access rights – is a channel which can restrict by IP addresses higher or lower risk than a channel which implements anti-DoS protection?

However, entry/exit analysis holds the promise of giving usable information which speaks meaningfully about the internal security posture of a codebase, and which is derived from less work than that required for a comprehensive manual analysis.

Future Work

Measurement of system attack surfaces could be improved by viewing the code at a level other than the function level, i.e., by looking inside functions and giving more weight to larger or more complex ones in analysis. This would be aided by the use of better tools which could trace the flow of execution through the code more accurately. It would also be useful to take into account which functions were actually compiled into the code in use, perhaps by comparing the full reachable code tree with symbol analysis of the resulting binary, or by replacing cflow with some tool which actually could read Makefiles.

Performing a data and channel analysis of entry and exit points would also provide a significant amount of information about the practicality of using entry/exit analysis to obtain a full picture of the system.

In addition, the problem of comparing codebases which perform the same overall function, but not the same subsets of that function, will need to be solved. For instance, if it were possible to determine what subset of the Cyrus and Courier codebases was dedicated to providing port listener functionality, then administrators could compare an analysis of inetd to just those subsets, and thereby determine whether the Cyrus and Courier implementations of that functionality were more or less risky than a third-party implementation.

Comparison of IMAP Server Software

I will finish with some brief words about the relative security of IMAP servers as a result of this analysis. My subjective impression that the Cyrus and Courier codebases are better designed than the UW codebase was corroborated by the entry/exit analysis results. In particular, the lack of internal privilege separation in UW-IMAP indicated that that codebase is not designed in a security-conscious way. In comparing Cyrus and Courier, I determine that Cyrus is better built, but overreaches significantly in terms of the number and diversity of features it offers, while Courier is designed in a very security-conscious manner, but implemented somewhat more strangely. My conclusion is that despite the administrative headaches it introduces, Courier is likely to be the best security risk when choosing one of these three products to act as an open source IMAP server.

Acknowledgements

I would like to thank Jeannette Wing and Dawn Song for enabling me to do this project, introducing me to the attack surfaces framework, and giving me advice and assistance along the way. I would also like to thank Pratyusa Manadhata for all his help with attack surfaces and entry/exit points, and Rob Siemborski for his explanations and corrections on the subject of IMAP implementations.

Availability

Because much of my entry/exit analysis was performed manually, there is no distributable software associated with this project. The project page at http://www.glassonion.org/projects/imap-attack/ contains a more detailed description of the methodology, along with scripts I used, and the detailed output of the analysis.

Author Information

Chaos Golubitsky first worked in system administration while an undergraduate at Swarthmore College. After earning a B.A. in Mathematics and Computer Science, she worked at the Harvard-Smithsonian Center for Astrophysics for three years, where she became interested in log monitoring. She returned to school for an M.S. in Information Security from Carnegie Mellon University, where she focused on practical analysis and improvement of system and software security.

References

[1] *Common Vulnerabilities and Exposures Database*, http://www.cve.mitre.org .

[2] *Courier-IMAP*, http://www.courier-mta.org/imap/ .

[3] *Cqual*, http://www.cs.umd.edu/~jfoster/cqual/ .

[4] *Cyrus IMAP Server*, http://asg.web.cmu.edu/cyrus/ .

[5] *Doxygen*, http://www.stack.nl/~dimitri/doxygen/ .

[6] *Exuberant Ctags*, http://ctags.sourceforge.net/ .

[7] *FreeBSD Ports: cflow*, http://www.freebsd.org/cgi/url.cgi?ports/devel/cflow/pkg-descr .

[8] *GNU Binutils*, http://www.gnu.org/software/binutils/ .

[9] *University of Washington IMAP*, http://www.washington.edu/imap/ .

[10] "UW-imapd fails to properly authenticate users when using CRAM-MD5," *Vulnerability Note VU#702777*, US-CERT, http://www.kb.cert.org/vuls/id/702777 , January, 2005.

[11] Howard, M., J. Pincus, and J. M. Wing, "Measuring Relative Attack Surfaces," *Proceedings of Workshop on Advanced Developments in Software and Systems Security*, August, 2003.

[12] Kamp, P.-H. and R. N. Watson, *Jails: Confining the Omnipotent Root*, Technical report, FreeBSD Project, http://docs.freebsd.org/44doc/papers/jail/jail.html .

[13] Manadhata, P., "Entry Points and Exit Points," Personal communication, 2005.

[14] Manadhata, P. and J. M. Wing, *Measuring a System's Attack Surface,* CMU-CS 04-102, Carnegie Mellon University, January, 2004.

[15] Mullet, D. and K. Mullet, *Managing IMAP*, O'Reilly Media, Inc., 2000.

Fast User-Mode Rootkit Scanner for the Enterprise

Yi-Min Wang and Doug Beck – Microsoft Research, Redmond

ABSTRACT

User-mode resource hiding through API interception and filtering is a well-known technique used by malware programs to achieve stealth. Although it is not as powerful as kernel-mode techniques, it is more portable and reliable and, as a result, widely used. In this paper, we describe the design and implementation of a fast scanner that uses a cross-view diff approach to detect all user-mode hiding Trojans and rootkits. We also present detection results from a large-scale enterprise deployment to demonstrate the effectiveness of the tool.

Introduction

The term "rootkit" generally refers to the class of stealth malware programs that hide "resources" from the operating system resource-enumeration APIs. For example, a rootkit may be used to hide critical executable files from anti-virus scanners, to hide critical Windows Registry entries from experienced system administrators using programs such as RegEdit, and to hide critical processes from average users running Windows Task Manager. By making critical resources "invisible" to the APIs and the system utilities that make use of such APIs, rootkits have a much better chance of evading detection and maintaining full control of infected machines for an extended period of time.

The techniques used by rootkits to achieve stealth can be broadly divided into two categories. Rootkits in the first category intercept resource-enumeration APIs in user mode or kernel mode, and remove selected entries before the results are returned to the API caller. Rootkits in the second category perform Direct Kernel Object Manipulation (DKOM) to remove selected resource entries from a cached list (such as the ActiveProcessList on Windows) that is designed specifically for answering resource queries and not critical to the actual functions of the resources or to the functioning of the operating system.

Cross-View Diff-Based Rootkit Detection

The traditional signature-based anti-virus approach cannot effectively deal with rootkit infections in the enterprise for three reasons. First, viruses usually have well-defined bundles and scope of impact, which can be analyzed in a lab to generate fixed signatures. In contrast, rootkits are merely "resource hiders" that can be used to hide any hacker tools, keyloggers, spyware programs, FTP servers, etc., so each rootkit infection can potentially involve a customized bundle with a different scope of impact.

Second, sophisticated hackers who attack large enterprise are less likely to use common tools or malware programs for which commercial anti-virus scanners already have signatures. Finally, while the major strength of anti-virus software is to detect and *automatically remove* known-bad malware programs without user intervention, corporate security organizations in large enterprises often need to investigate every rootkit infection case to assess potential damages and prevent future infections; automatic removal is often not desirable.

We previously proposed a non-signature, diff-based approach to rootkit detection, called *Ghost-Buster* [WVR+04]. The basic idea is to get "the lie" from inside the box, get "the truth" from outside the box, and take a diff to detect hidden resources. Specifically, we get "the lie" by enumerating files and Registry entries through infected APIs inside the operating system. Then we boot into a clean CD and scan the files and Registry on the infected drive as a data drive. Since the rootkit is not running, we obtain "the truth" that includes the resources that the rootkit was trying to hide so that the diff between "the lie" and "the truth" will reveal precisely those hidden entries. Such a diff-based approach essentially turns the hiding behavior into its own detection mechanism and turns one of the most difficult anti-malware problems into one of the easiest problems to solve.

Fast User-Mode Rootkit Scanner for the Enterprise

Although this CD-boot-based solution can cover a broad range of rootkits, no matter how they are operating in user mode or kernel mode, it is inconvenient, requires user cooperation, and is difficult to deploy on an enterprise scale as a scanner. Since the statistics from a major Product Support Service (PSS) organization indicates that *user-mode rootkits* account for over 90% of the reported enterprise rootkit cases, it is desirable to have a scalable rootkit scanner that can be deployed in the enterprise to detect all user-mode rootkits, which intercept and filter resource API calls in the address space of each user-mode process [YH03, YN04].

We have developed such a rootkit scanner for Windows platforms. It is based on the key observation that, when considering only user-mode rootkits, "the truth" can be obtained from *the lowest level of user mode* by properly preparing the call stack parameters and using a few lines of assembly code to directly invoke the transition into the kernel, without going through the regular user-mode Win32 API code. In our current implementation, we use the difference to detect hidden processes and to detect hidden hooks to Auto-Start Extensibility Points (ASEPs), which are those Registry locations most frequently attacked by spyware, Trojans, and rootkits based on an extensive study [WRV+04, WBV+05]. If any hidden processes or ASEP hooks are detected, we then look for potentially hidden files associated with those hidden entries. This allows us to detect user-mode rootkit infections in a few seconds, without requiring a reboot or any user participation.

Implementation

Figure 1 (a) and (b) illustrates how several real-world Trojans and rootkits hook into the Registry and process enumeration API calling chains, respectively, to hide their resources. Urbin and Mersting are Trojan DLLs that make modifications at the highest level by altering the per-process Import Address Table (IAT) entries of the Registry enumeration APIs to point to their Trojan functions (an IAT contains pointers to functions exported by loaded DLLs [HB05]). In contrast, Vanquish directly modifies the loaded, in-memory API code to interject its code. Both techniques cause the Trojan functions to appear in the call stack trace of a kernel or user-mode debugging session.

To achieve better stealth, Aphex and Hacker Defender modify the in-memory API code with a jump to the Trojan code along with a Trojan code jump back to the next instruction after the API detour [HB99]; the Trojan code modifies the return address on the stack to cause its code to be executed in the return path. The only difference is that Aphex modifies the RegEnumValue API code inside Apvapi32.dll (denoted by Advapi32!RegEnumValue), while Hacker Defender modifies the lower-level NtEnumerateKey API exported by NtDll.dll. YYT rootkit operates very similarly to Hacker Defender. ProBot SE in Figure 1(a) and FU in Figure 1(b) are kernel-mode stealth programs that cannot be detected by the scanner described in this paper.

Our tool performs the following steps to obtain "the truth" from underneath all the user-mode Trojans and rootkits shown in Figure 1.

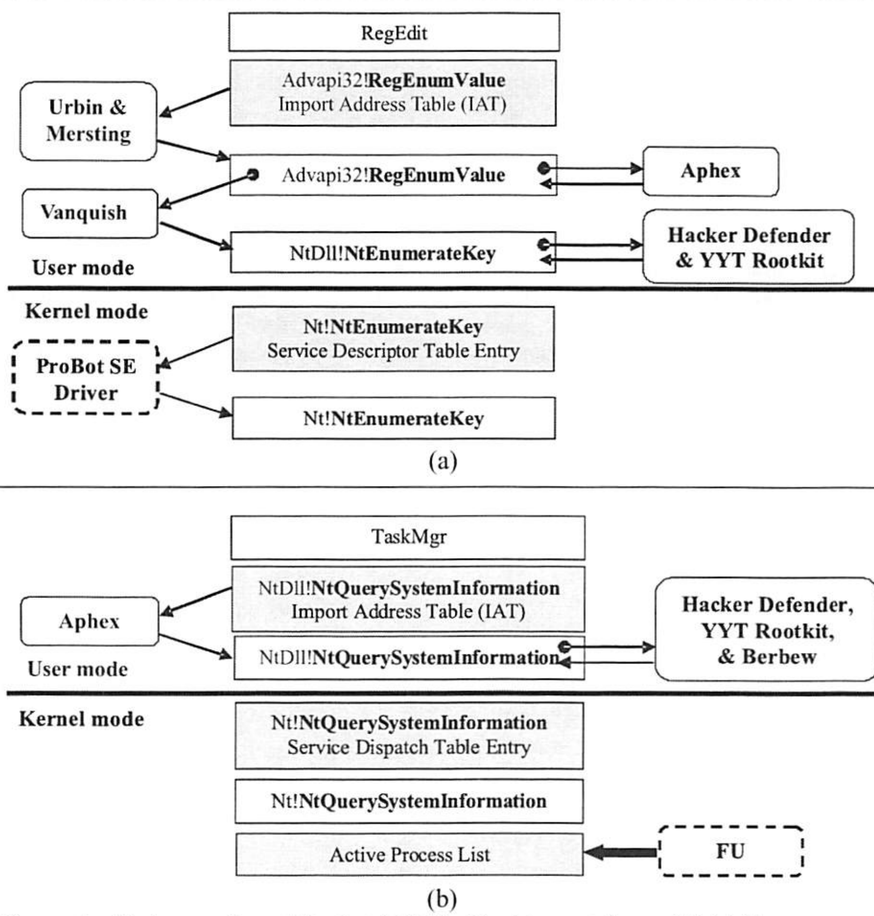


Figure 1: Trojans and rootkits that (a) hide Registry entries and (b) hide processes.

1. Set up the user-mode stack with the parameters required by the operating system; this is easily achieved by creating a function with a signature that exactly matches the desired Native API, such as *NtDll!NtEnumerateKey()* and *NtDll!NtQuerySystemInformation()*;
2. Populate the EAX register with the index that indicates to the operating system which system function you wish to call;
3. Populate the EDX register with a pointer to the user mode's stack parameters;
4. Execute an "int 2e" instruction to signal a kernel-mode transition;
5. Return to the caller.

The previous steps basically describe what the code in *NtDll.dll* does when calling into the operating system. Given that the steps require direct manipulation of X86 registers, a portion of the code is written in assembly – this is easily obtained by disassembling *NtDll.dll* and searching for the desired function by name. Below is an example of the *NtQuerySystemInformation* call for retrieving "the truth" of the list of processes:

```
__declspec(naked)
NTSTATUS
NTAPI
MyNtQuerySystemInformation(
    SYSTEM_INFORMATION_CLASS
                    SystemInformationClass,
    PVOID SystemInformation,
    ULONG SystemInformationLength,
    PULONG ReturnLength)
{
    __asm
    {
            mov eax, 0xAD
            lea edx, [esp+0x4]
            int 2eh
            ret 10h
    }
}
```

Several important points need to be made regarding the code above. First, notice the `__declspec (naked)` compiler directive. This prevents the compiler from generating prolog code for the function in order to ensure that the user-mode stack is in the right form at the time of a kernel-mode transition. Second, the function is labeled as `NTAPI` to ensure that the right C calling convention is used (in this case `__stdcall`). Third, the value that is moved into the EAX register is the index into a kernel-mode function dispatch table that tells the operating system which function to call: this value is unique to the function and varies from version to version of the operating system. Fourth, the "int 2eh" instruction sends a signal to the operating system to tell it to initiate a kernel-mode transition. Finally, the parameters exactly match the original query API because the operating system will pass them directly to the kernel-mode API.

Rootkits & Trojans	Hidden ASEP Hooks Detected
Urbin	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs → msvsres.dll
Mersting	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs → kbddfl.dll
YYT Rootkit	HKLM\SYSTEM\CurrentControlSet\Services\NPF → npf.sys HKLM\SYSTEM\CurrentControlSet\Services\TSSERVER → comine.exe HKLM\SYSTEM\CurrentControlSet\Services\Udfs
Hacker Defender 1.0	HKLM\SYSTEM\CurrentControlSet\Services\HackerDefender100 → hxdef100.exe HKLM\SYSTEM\CurrentControlSet\Services\HackerDefenderDrv100 → hxdefdrv.sys
Aphex	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run → <user defined name>.exe
Vanquish	HKLM\SYSTEM\CurrentControlSet\Services\Vanquish → vanquish.exe

(a)

Rootkits & Trojans	Hidden Processes Detected
Berbew	<random name>.exe
YYT Rootkit	comine.exe
Hacker Defender 1.0	hxdef100.exe, and any other processes with names matching the patterns specified in hxdef100.ini
Aphex	By default, any process with a "~~"-prefixed name (which is configurable)

(b)

Figure 2: Hidden resources detected: (a) hidden Registry ASEP hooks; (b) hidden processes.

Once the tool obtains "the truth" and then uses a regular Win32 API call to obtain "the lie", it compares the two scans and declares as hidden resources those that appear only in "the truth". It is possible to see false positives due to the creation or deletion (depending on the order the scans are performed) of resources in the time window between the scans. In practice this is usually not an issue. Also, this is easily mitigated by performing successive scans and taking the intersection of the missing resource sets to create the final result set.

Experimental Results

Results from Lab Tests

We have tested the tool in our lab against the seven user-mode rootkits and Trojans shown in Figure 1. Although the implementations of these malware programs are quite different, our tool was able to efficiently and effectively detect all of them in a uniform way. Figure 2(a) and (b) show the detected hidden ASEP hooks and hidden processes, respectively, for each malware. The Urbin, Mersting, Berbew, and YYT Rootkit samples were captured from the wild, while the Hacker Defender, Aphex, and Vanquish samples were downloaded from the Web. The "AppInit_DLLs" ASEP allows auto-loading of one or more DLLs into every Windows-based application that is running in the current log-on session [AID]; the "Services" ASEP allows installations of always-running services and drivers; the "Run" ASEP allows additional processes to be auto-started near log-in time.

Results from Actual Deployment

We have deployed the tool on over 200,000 desktop and server machines. The executable file is copied from a central machine to each target machine at scan time; the scan results are reported back to the central machine, and the executable file is removed. Figure 3 gives some examples of detected infections. They can be broadly classified into two categories: hiding Trojans and full-fledged rootkits. Figures 3 (a), (b), and (c) show three types of hiding Trojans: they were most likely installed by malicious Web servers that exploit visiting browsers' vulnerabilities [WBJ+05]. The Trojans in Figure 3 (a) hide their hooks to the "AppInit_DLLs" ASEP, while the Trojans in Figure 3 (b) and (c) hide their randomly-named processes.

Figure 3 (d), (e), (f) show three cases of infections with full-fledged rootkits. We make the following observations: first, it is common for rootkits to create malware programs that have the same or similar filenames as some system programs but reside in a different directory; for example, one of the lsass.exe processes in Figure 3 (f) was instantiated from the lsass.exe malware program located in the "drivers" directory. Second, full-fledged rootkits tend to hook the "Services" ASEP to install services and drivers and they tend to hide multiple ASEP hooks and processes.

Rootkit Investigation Tool

Once a rootkit-infected machine is identified in an enterprise, it is important to investigate the hacker's intention and the damage that has been done, without disturbing the malware because some are designed to erase all traces of themselves once they realize that they have been detected. It is therefore highly desirable to have a tool that allows such non-intrusive investigations.

We have observed that configurable rootkits, such as the most popular Windows rootkit "Hacker Defender," typically support the notion of "root processes". A root process is not infected by the rootkit and can see "the truth". It is provided for the convenience of the rootkit users. For example, it can be very awkward for hackers if the resources are hidden from their tools

Hidden ASEP Hook: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\
Value: "AppInit_DLLs" Data: "C:\WINDOWS\system32\log.dll"
Hidden ASEP Hook: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\
Value: "AppInit_DLLs" Data: "C:\WINDOWS\System32\winpgfd.dll"
Hidden ASEP Hook: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\
Value: "AppInit_DLLs" Data: "C:\WINDOWS\System32\winpgfd.dll"

(a)

Hidden Procs: \Device\HarddiskVolume1\WINDOWS\system32\Hbkqjd32.exe
Hidden Procs: \Device\HarddiskVolume2\WINDOWS\system32\Mdnpdf32.exe
Hidden Procs: \Device\HarddiskVolume1\WINDOWS\System32\Ppnlan32.exe
Hidden Procs: \Device\HarddiskVolume1\WINDOWS\system32\Ljoocp32.exe
Hidden Procs: \Device\HarddiskVolume1\WINDOWS\system32\Gnaeplam.exe

(b)

Hidden Procs: \Device\HarddiskVolume1\WINDOWS\System32\elitesig32.exe
Hidden Procs: \Device\HarddiskVolume1\WINDOWS\system32\elitegct32.exe
Hidden Procs: \Device\HarddiskVolume2\WINDOWS\system32\eliteyzx32.exe
Hidden Procs: \Device\HarddiskVolume1\WINNT\system32\eliteohl32.exe
Hidden Procs: \Device\HarddiskVolume2\WINNT\system32\elitemfu32.exe
Hidden Procs: \Device\HarddiskVolume2\WINNT\system32\eliteaee32.exe

(c)

Figures 3a-3c: Actual infection cases.

as well; when a rootkit is used to hide a spyware browser add-on executable file, it gets really tricky if the file is also hidden from the browser process that is supposed to load it. With "root process" support, the hacker tools and the browser can be declared as root processes to ensure smooth operation, while the resources are still hidden from all the other processes, system utilities, and anti-malware scanners. In the case of Hacker Defender, "root processes" are listed in a configuration file via filenames specified with regular expressions.

We have developed a technique to take advantage of the support for root processes to allow non-intrusive investigations. It works as follows [WB05]. First, the fast user-mode rootkit scanner is used to identify hidden processes. Very often, one or more of the hidden processes are also root processes (because these are the most critical processes for the infection). Suppose process "foo.exe" is a detected hidden, root process. In the second step, we make a copy of the command-window program "cmd.exe", rename it to "foo.exe", and launch it through "start foo.exe". Now we have a command window that is a root process that can see all previously hidden resources. In particular, the "dir" command from this window can see all hidden files, including the Hacker Defender configuration file which reveals critical information about the hacker's intention.

```
Hidden processes detected: 2
    PID  Name
    2168 \Device\HarddiskVolume1\WINDOWS\system32\scrss.exe
    2460 \Device\HarddiskVolume1\WINDOWS\system32\taskmgnr.exe
Hidden service keys detected: 2
Key: HKEY_LOCAL_MACHINESYSTEM\CurrentControlSet\Services\scrss
       ImagePath: C:\WINDOWS\system32\scrss.exe
Key: HKEY_LOCAL_MACHINESYSTEM\CurrentControlSet\Services\tskmgr
       ImagePath: C:\WINDOWS\system32\taskmgnr.exe
```

(d)

```
Hidden processes detected: 11
    PID   Name
    436   \Device\HarddiskVolume2\WINDOWS\system32\smss.exe
    564   \Device\HarddiskVolume2\WINDOWS\system32\lsass.exe
    1396  \Device\HarddiskVolume2\WINDOWS\system32\svchosts.exe
    1428  \Device\HarddiskVolume2\WINDOWS\system32\csschk.exe
    1744  \Device\HarddiskVolume2\WINDOWS\system32\DNTUS26.EXE
    1976  \Device\HarddiskVolume2\WINDOWS\system32\ShellExt\_\tmp\IPSSvc.exe
    248   \Device\HarddiskVolume2\ SYSTEM~1\_system\lsass.exe
    308   \Device\HarddiskVolume2\WINDOWS\smss.exe
    364   \Device\HarddiskVolume2\ SYSTEM~1\_system\system\ioFTPD.exe
    2052  \Device\HarddiskVolume2\ SYSTEM~1\_system\lsass.exe
    2164  \Device\HarddiskVolume2\ SYSTEM~1\_system\bot\eggdrop.exe
Hidden service keys detected: 2
Key: HKEY_LOCAL_MACHINESYSTEM\CurrentControlSet\Services\IPSdrv
       ImagePath: \??\c:\windows\system32\shellext\_\tmp\IPSdrv.sys
Key: HKEY_LOCAL_MACHINESYSTEM\CurrentControlSet\Services\NetSecc
       ImagePath: c:\windows\system32\shellext\_\tmp\ipssvc.exe naslib.dll
```

(e)

```
Hidden processes detected: 4
    PID   Name
    556   \Device\HarddiskVolume2\WINDOWS\system32\services.exe
    568   \Device\HarddiskVolume2\WINDOWS\system32\lsass.exe
    1768  \Device\HarddiskVolume2\WINDOWS\java\lspool.exe
    484   \Device\HarddiskVolume2\WINDOWS\system32\drivers\lsass.exe
Hidden service keys detected: 2
Key: HKEY_LOCAL_MACHINESYSTEM\CurrentControlSet\Services\Lspool
            ImagePath: c:\windows\java\lspool.exe
Key: HKEY_LOCAL_MACHINESYSTEM\CurrentControlSet\Services\r_server
            ImagePath: c:\windows\system32\drivers\lsass.exe /service
```

(f)

Figures 3d-3f: Actual infection cases.

In the third step, we launch Task Manager, RegEdit, anti-virus scanner, anti-spyware scanner, etc. from this command window. Since this root process is not infected it cannot infect its child processes; as a result, all these utilities are now running as root processes that can see all previously hidden resources. After the investigation, malware processes, Registry entries, and files can all be terminated/deleted through these utilities.

Related Work

There are two different approaches to rootkit detection. The first approach targets the hiding mechanism by, for example, detecting the presence of API interceptions [YI, ZVI, YK, YKS, YV04]. It has at least two disadvantages: first, it cannot catch rootkit programs that do not use the targeted mechanism; second, it may catch as false positives legitimate uses of API interceptions for in-memory software patching, fault-tolerance wrappers, security wrappers, etc. The second approach targets the hiding behavior by detecting any discrepancies between "the truth" and "the lie". For example, comparing the output of "ls" and "echo *" can detect an infected "ls" program [B99]. Our user-mode rootkit detector belongs to the second category.

There is a subtle but important difference between the "cross-view diff" used in our proposal and the more common "cross-time diff" used in Tripwire [KS94] and the Strider Troubleshooter [WVS03, WVD+03]. The goal of a cross-time diff is to capture changes made to persistent state by essentially comparing snapshots from two different points in time (one before the changes and one after). In contrast, the goal of a cross-view diff is to detect hiding behavior by comparing two snapshots of the same state at exactly the same point in time, but from two different points of view (one through the infected path and one not). Cross-time diff is a more general approach for capturing a broader range of malware programs, hiding or not; the downside is that it typically includes a significant number of false positives stemming from legitimate changes and thus requires additional noise filtering, which has a negative impact on usability. In contrast, cross-view diff targets only hiding malware and usually has zero or very few false positives because legitimate programs rarely hide.

Ideally, "the truth" should be obtained from "outside the box" to eliminate the possibility of any malware intervention. The WinPE-based GhostBuster tool [WBV+05] took such an approach to obtain "the truth" of the file system and Registry. The PCI-add-in card described in the Copilot paper [PFM+04] or the Myrinet NIC described in the Bookdoors paper [BNG+04] can be used to obtain "the truth" of the process list through Direct Memory Access (DMA) without the knowledge or intervention of the potentially infected OS. Instead of targeting comprehensiveness, our user-mode rootkit detector targets efficiency, scalability, and ease of use with good coverage.

In response to the increasing popularity of stealth techniques among Windows malware, several rootkit detection tools have been released in recent months, including RootkitRevealer from Sysinternals, Blacklight Rootkit Eliminator from F-Secure, and IceSword from Xfocus.net. RootkitRevealer uses the same cross-view diff technique described in our previous paper on Inside-the-box GhostBuster [WBV+05]: it performs high- and low-level scans and reports the differences between these scans. For the file scans, it performs two low-level scans by reading the NTFS Master File Table and the NTFS on-disk directory index structures. For the low-level Registry scan, the tool reads the raw Registry hive files. A file discrepancy is reported if a file does not appear in all three scans. A Registry discrepancy is reported if the data, length, or type of a Registry value differs or if an entry is missing.

Blacklight is designed to detect hidden processes and files via a kernel-mode driver. In addition to running as a standalone process, it incorporates detection evasion technology: it may perform its scans through the Windows Explorer process. IceSword uses kernel-mode technology to detect hidden processes, hidden ports, hidden services, hidden auto-start programs, hidden files, hidden Browser Helper Objects, and hidden Registry entries. It monitors process creation and deletion and is able to disable filter drivers that prevent file creation and deletion. In order to achieve its functionality, the program loads a kernel-mode driver and then disables kernel-level debugging. Unlike the other rootkit tools, it has anti-rootkit attack technology to keep it from being disabled by malicious software. It achieves this by trapping keyboard strokes and requiring that the user hit Ctrl+Alt+D in order to put the program into a mode where it may be shutdown.

Similar rootkit problems exist on the Linux/UNIX platforms as well [PFM+04, YKS, YC, YW98, B99, YA03]. (In fact, the term "rootkit" originated from the root privilege concept on UNIX platforms.) A common technique used by Linux/UNIX rootkits to hide resources is to intercept system calls to the kernel via a Loadable Kernel Module (LKM) [ZK, YJ, J01]. For example, some rootkits are known to hook read, write, close, and the getdents (get directory entries) system calls. More advanced rootkits can directly patch the kernel in memory [YC98, YL01]. We discussed cross-view diff-based hidden resource detection for Linux/UNIX platforms in our previous paper [WBV+05].

As a final note, most of today's Windows rootkits do not modify OS files or memory image; rather, they "extend" the OS through ASEP hooking in a way that is indistinguishable from many other good software programs that also extend the OS. Therefore, it is difficult to apply the genuinity tests and software-based attestation techniques that detect deviations from a known-good hash of a well-defined OS memory range [KJ03, SPDK04]. On the other hand, these

techniques can detect both hiding and non-hiding malware programs that modify the OS and are complementary to the cross-view diff approach.

Summary

User-mode rootkits are popular because they are more portable and reliable than kernel-mode rootkits. We have shown that there is a quick and easy way to detect all user-mode rootkits: by performing a cross-view diff between a high-level infected scan above rootkit interception and a low-level clean scan below the interception, our tool can precisely detect hidden Registry entries and processes within a few seconds. The simplicity and efficiency make it an attractive tool for scalable deployment in large enterprises to provide protection against new or customized rootkits that escape common signature-based anti-malware scanning. Detection results from actual deployment suggested that hiding Trojans, most likely installed through malicious Web sites, may be an even more serious concern than rootkits in terms of prevalence.

Author Information

Yi-Min Wang manages the Cybersecurity and Systems Management Research Group and leads the Strider project at Microsoft Research, Redmond. He received his Ph.D. in Electrical and Computer Engineering from University of Illinois at Urbana-Champaign in 1993, worked at AT&T Bell Labs from 1993 to 1997, and joined Microsoft in 1998. His research interests include security, systems management, dependability, home networking, and distributed systems.

Doug Beck is a senior developer at Microsoft where he has worked for the past six years. During his career at Microsoft Doug has focused on developing Systems Management software. He received a Ph.D. in Theoretical Physical Chemistry from the University of Washington in 1996 where he developed simulation software for solving partial differential equations as part of his research. Doug is currently working at Microsoft Research and can be reached at Doug.Beck@microsoft.com .

References

[AID] Working with the AppInit_DLLs registry value, http://support.microsoft.com/kb/q197571/.

[AS] Microsoft Windows Anti-Spyware, http://www.microsoft.com/spyware .

[B99] Brumley, D., "Invisible Intruders: Rootkits In Practice," *;login:*, http://www.usenix.org/publications/login/1999-9/features/rootkits.html, 1999.

[BNG+04] Bohra, A., I. Neamtiu, P. Gallard, F. Sultan, and L. Iftode, "Remote Repair of Operating System State Using Backdoors," *Proc. Int. Conf. on Autonomic Computing (ICAC)*, pp. 256-263, May, 2004.

[HB99] Hunt, Galen and Doug Brubacher, "Detours: Binary Interception of Win32 Functions," *Proc. the Third Usenix Windows NT Symposium*, pp. 135-143, http://research.microsoft.com/sn/detours/ , July, 1999.

[HB05] Hoglund, G. and J. Butler, *Rootkits: Subverting The Windows Kernel*, Addison-Wesley, 2005.

[J01] Jones, K., "Loadable kernel modules," *;login:*, http://www.usenix.org/publications/login/2001-11/pdfs/jones2.pdf, Nov., 2001.

[KJ03] Kennell, Rick and Leah H. Jamieson, "Establishing the Genuinity of Remote Computer Systems," *Proc. USENIX Security Symposium*, August, 2003.

[KS94] Kim, G. H. and E. H. Spafford, "The Design and Implementation of Tripwire: A File System Integrity Checker," *Proc. of the Second ACM Conf. on Computer and Communications Security*, pp. 18-29, Nov., 1994.

[MSRT] Windows Malicious Software Removal Tool, http://www.microsoft.com/security/malwareremove/ .

[PFM+04] Petroni, Jr., Nick L., Timothy Fraser, Jesus Molina, and William A. Arbaugh, "Copilot – a Coprocessor-based Kernel Runtime Integrity Monitor," *Proc. Usenix Security Symposium*, Aug., 2004.

[SPDK04] Seshadri, A., A. Perrig, L. van Doorn, and P. Khosla, "SWATT: SoftWare-based ATTestation for Embedded Devices," *Proc. IEEE Symp. on Security and Privacy*, May, 2004.

[WB05] Wang, Yi-Min and Doug Beck, "How to 'Root' a Rootkit That Supports Root Processes Using Strider GhostBuster Enterprise Scanner," *Microsoft Research Technical Report MSR-TR-2005-21*, February 11, 2005.

[WBJ+05] Wang, Yi-Min, Doug Beck, Xuxian Jiang, and Roussi Roussev, "Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities," *Microsoft Research Technical Report MSR-TR-2005-72*, August, 2005.

[WBV+05] Wang, Yi-Min, Doug Beck, Binh Vo, Roussi Roussev, and Chad Verbowski, "Detecting Stealth Software with Strider GhostBuster," *Proc. DSN*, June, 2005.

[WRV+04] Wang, Yi-Min, Roussi Roussev, Chad Verbowski, and Aaron Johnson, "Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management," *Proc. Usenix LISA*, Nov., 2004.

[WVD+03] Wang, Yi-Min, et al., "STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support," *Proc. Usenix LISA*, pp. 159-171, October, 2003.

[WVR+04] Wang, Yi-Min, Binh Vo, Roussi Roussev, Chad Verbowski, and Aaron Johnson, "Strider GhostBuster: Why It's A Bad Idea For Stealth Software To Hide Files," *Microsoft Research Technical Report MSR-TR-2004-71*, July, 2004.

[WVS03] Wang, Yi-Min, Chad Verbowski, and Daniel R. Simon, "Persistent-state Checkpoint Comparison for Troubleshooting Configuration Failures," *Proc. IEEE DSN*, June, 2003.

[YA03] Chuvakin, A. "An Overview of UNIX Rootkits," iALERT White Paper, iDefense Labs, http://www.megasecurity.org/papers/Rootkits.pdf, February, 2003.

[YC] The chkrootkit tool, http://www.chkrootkit.org/ .

[YC98] Cesare, Silvio "Runtime kernel kmem patching," http://vx.netlux.org/lib/vsc07.html , Nov., 1998.

[YH03] "How to become unseen on Windows NT," http://rootkit.host.sk/knowhow/hidingen.txt , May 8, 2003.

[YI] Ivanov, Ivo, "API hooking revealed," http://www.codeproject.com/system/hooksys.asp .

[YJ] Jones, A. R., "A Review of Loadable Kernel Modules," http://www.giac.org/practical/gsec/Andrew_Jones_GSEC.pdf .

[YK] Keong, Tan Chew, "ApiHookCheck Version 1.01," http://www.security.org.sg/code/apihookcheck.html , April 15, 2004.

[YKS] KSTAT – Kernel Security Therapy Anti-Trolls, http://s0ftpj.org/en/tools.html .

[YL01] "Linux on-the-fly kernel patching without LKM," http://www.phrack.org/phrack/58/p58-0x07, *Phrack Magazine*, Dec., 2001.

[YN04] "NTIllusion – A portable Win32 userland rootkit.txt," *Phrack Magazine*, July 13, 2004.

[YV04] VICE – Catch the hookers! http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-butler/bh-us-04-butler.pdf .

[YW98] "Weakening the Linux Kernel," Phrack Magazine, http://www.phrack.org/phrack/52/P52-18, Jan., 1998.

[ZK] Knark LKM-rootkit, http://www.sans.org/resources/idfaq/knark.php .

[ZVI] Vice, http://www.rootkit.com/project.php?id=20 .

Configuration Tools: Working Together

Paul Anderson and Edmund Smith – University of Edinburgh

ABSTRACT

Since the LISA conferences began, the character of a typical "large installation" has changed greatly. Most large sites tended to consist of a comparatively small number of hand-crafted "servers" supporting a larger number of very similar "clients" (which would usually be configured with the aid of some automatic tool). A modern large site involves a more complex mesh of services, often with demanding requirements for completely automatic reconfiguration of entire services to provide fault-tolerance. As these changes have happened however, the tools available to provide configuration management for a site have not evolved to keep pace with these new challenges. This paper looks at some of the reasons why configuration tools have failed to move forward, and presents some suggestions for enabling the state of the art to advance.

Background and Motivation

Configuration Tools have been an important theme at LISA for many years, and most conferences include one or more papers in this area. Despite increasing recognition of the importance of the configuration problem, there remains both a lack of conceptual commonality and a lack of progressive innovation in the area.

No significant standards have so far emerged, and new tools often merely espouse variations on existing approaches, frequently using both completely new specifications and code. For example, ten years separate the initial description of LCFG [5] and the presentation of Newfig [12], yet there is no clear evidence of conceptual progress. There is little or no attempt to create standards in a way which would reduce the barrier to new development or enable a greater shared understanding of the configuration problem.

Within the configuration management community, a great deal of ongoing discussion revolves around this apparent failure to either successfully disseminate the concepts underpinning existing tools, and the experiences gained from those tools, or to realise them in tool implementations suitable for a wider audience.

Furthermore, despite the slow rate of progress, no sign of convergence between tools is apparent. There are essentially no standards in this area, with each tool being not only entirely coded from scratch, but also unable to interoperate with other tools, or share configuration data with them. Although the Configuration Description, Deployment and Lifecycle Management (CDDLM) working group of the Global Grid Forum (GGF) has published a standard interface for configuration tools on grid fabrics, their focus is upon exposing a web-service interface to the underlying configuration system, which falls somewhat wide of the mark in addressing the current problems faced by system administrators.

This absence of standard tools has lead to many large sites developing their own tools. To create a functional configuration management system in its entirety, however, is a significant undertaking, and even those sites which have been able to invest sufficient resources in a system to make it sustainable have been unable to gain a wider community of users. It seems likely that in many cases, the sheer difficulty of developing and maintaining a large monolithic system (often written by people with a shortage of time and no background in software development) has limited both the functionality and the portability of tools.

It is surprising indeed, given the importance of this area, that we find ourselves typically unable to recommend any current system to interested users.

Levels of Configuration

Perhaps because of the above difficulties, most existing tools deal with configuration specifications at a very primitive level; they involve "files" and "permissions", rather than "high-level" concepts such as "services", and the relationships between nodes – Figure 1 shows some typical statements at increasingly higher levels of abstraction.

Ultimately, the requirements for a service are always expressed in high-level terms and most tools require these to be manually translated first into some lower-level requirements. As systems become more complex, this process is an increasing source of errors. Manual intervention is also unacceptable for autonomic systems which must reconfigure automatically in response to demand and failure. A future generation of configuration tools will need to be able to accept high-level requirement specifications, and to reason in much more sophisticated ways in order to determine the appropriate details. This is an inevitable consequence of the growing complexity of the modern installation, and the fact that the number of administrators a site needs does not scale linearly with the number of machines managed.

There is an analogy with computer programming here: early languages provided very little abstraction from the underlying machine, but as the size and

complexity of systems has grown, so has the separation from the hardware operations. In a similar way, configuration tools must adapt to aid administrators to manage the complexity of their systems.

Low Level

"Copy this disk image onto these machines"

↓

"Put these files on these machines"

↓

"Put this line in sendmail.cf on this machine"

↓

"Configure machine X as a mail server"

↓

"Configure machine X as a mail server for this cluster" (and the clients will automatically be configured to match)

↓

"Configure any suitable machine as a mail server for this cluster" (and the clients will automatically be configured to match)

↓

Configure enough mail servers to guarantee an SMTP response time of X seconds

High Level

Figure 1: Levels of configuration specification.

The Configuration Tool Zoo

There are a plethora of tools available which purport to address some aspect of the configuration management problem. Many of these are described in more detail in surveys such as [6]. However, we present the following examples as typical, in that they demonstrate a variety of the most common approaches. In particular we note that:

- The tools have different *lexicons* – for example, there is no common agreement on how to specify the "mail relay" for a particular client.
- They use different languages and file formats.
- Their deployment operations provide different guarantees regarding pre- and post-conditions.

In general, these tools are also intended for direct manual use and present no intermediate interfaces between the human-generated specification, and the direct manipulation of the target machine.

Arusha

Arusha [11] uses an XML-based source language for describing configuration information. It is intended to facilitate sharing of configuration information between sites.

LCFG

LCFG [5, 7, 2] has a site-wide repository of declarative configuration data which is compiled into XML descriptions for transmission to the individual hosts. Clients execute *components*, each of which takes its specification from a corresponding section of the XML *profile*.

Quattor

Quattor [3] has a similar architecture to LCFG, differing principally in the exact forms of the components and the capabilities of the central compiler. Both Quattor and LCFG have been used successfully to specify "complete" configurations for large, complex sites.

BCFG

BCFG [9] is a new tool from Argonne National Laboratories, based upon a highly centralised model of configuration construction. Complete configuration file sets are computed centrally for each client (together with attribute data), before being distributed in an XML envelope.

Cfengine

Cfengine [8] is probably the most widely-used "configuration tool," but perhaps the most difficult to fit into a common model. It is frequently used to specify only partial configuration information, and its specification language is less well suited to manipulation by independent programs.

The CDDLM Standard

The CDDLM standard proposal [1] is interesting in that it represents the first formal attempt at a standard in this area that we are aware of. It is a proposed international standard which provides a tool-independent specification for the transmission of configuration information. The intention is to provide complete configuration control, aimed as it is at the management of grids – a highly federated environment requiring autonomic operation.

A Common Model

Despite the range of views and models set out earlier, there are in fact two distinct capabilities a modern system must provide:

1. Deciding what configuration a particular node should have.
2. Creating the desired configuration on a node.

This is the difference between, for example, deciding that a machine must have a package installed on it, and actually installing that package. The fundamental idea is to separate these two distinct roles into separate systems: systems that communicate through a configuration description. For the sake of brevity, in future we will call a tool able to decide what the configuration of a node should be a *configuration management system*, and a tool capable of creating a configuration on a node a *deployment engine*.

Writing a deployment engine requires an enormous amount of code, and keeping it current requires continuous maintenance. The systems we are asked to manage were not typically designed for neither bulk nor automated maintenance. There are a myriad of file formats, a slew of ways to restart a service, endless configuration files and preconditions. Each of these

will typically find a place somewhere within the deployment engine.

At present, each configuration tool author begins by writing a deployment engine, and many deployment engines exist (albeit inside a configuration tool, and not accessible independently). Once the deployment engine is complete, whatever time and resources remain can be devoted to configuration management. Unfortunately, as we mentioned earlier, the amount of time and resources remaining tends to be small, and configuration management has remained in a relatively primitive state for many years. At present, the only mechanism typically available is the ability to group nodes and configure them as a unit.

An overview of some existing configuration management operations is given in "Managing Configuration Data" together with a discussion of proposed alternatives. We argue that the core of the difficulty in advancing the state of the art is that each current project must include both a deployment engine and a configuration management tool. Whilst deployment engines are well-understood and well-developed, they represent such a large amount of work that must be performed before configuration management can begin that no time is available to develop advanced configuration management systems. This is despite the fact that it is not our inability to deploy a configuration on a node that makes a large modern installation so hard to manage, but that the amount of information involved is too large to be managed in an ad-hoc fashion.

By separating out the deployment engine from the configuration management tool, we make it possible to implement one without implementing the other. We believe that this could lead to a rapid advance in the sophistication of management tools. In addition, we believe that many of the existing configuration tools discussed previously could be released as deployment engines relatively easily, as their configuration management functions are both small and peripheral, so one might imagine easy to isolate. This is certainly true for LCFG, for example.

A Document Interface to Deployment Engines

In this paper we aim to convince the reader that if deployment engines conformed to a minimal configuration description standard, it would allow the independent development of configuration management tools. This appears the only plausible way to relieve the current stagnation in configuration tool development, and enable the creation of a new generation of configuration tool, capable of managing a modern network.

This section presents one possible interface to deployment engines, which we believe is most usefully realised as a document. This is similar to the familiar tool chain for program development (e.g., autoconf, automake, make, gcc, ld, etc.) More important than the format of the document is what such a document represents:

A *configuration description* specifies a complete, unambiguous and instantaneous configuration for the target machine or system.

- *Complete* means that everything the deployment engine is capable of specifying is given a value.
- *Unambiguous* means that no further logic is required to determine the parameters of the deployment engine. There are no references, no late binding, nor any database queries required.
- *Instantaneous* means that the specification is a snapshot of a node's state that the engine should be working towards. There is no time dependency stated.

Our proposed document format is XML-based, simply on the grounds that XML is a widely-used and convenient way of representing structured data such as configuration parameters. Most languages provide XML parsers and processing libraries which simplify the development of cross-platform libraries and tools. There is no suggestion here that tools should provide users with an XML interface, only that it is a convenient format for exchanging data between a management tool and a deployment engine.

Entries

```
<entry name="foo">
 <value>bar</value>
 <origin>
  <file name="foo.cfg" line="132"/>
 </origin>
</entry>
<entry>
 <value>baz</value>
 <origin>
  <file name="anon.cfg" line="211"/>
 </origin>
</entry>
```

An entry represents a data item. An entry may be either named or unnamed. When named, the entry represents a key-value pair, the most common method of storage for configuration data (e.g., [10, 5, 9, 3]). The entry's name attribute is the key, and the value is inside the value element (which must be text – there may not be any nested elements.) In the example, the key "foo" was bound to the value "bar". An entry may also be unnamed, for example for use as a member of a list of values. The origin element is optional, specifying a list of files and lines which were used to determine this value. It is purely to enable a deployment engine to give meaningful errors when the files the user edited were passed to a different tool.

Structs

```
<struct name="sudoers">
 <entry>
  <value>joe</value>
 </entry>
 <entry name="admin">
  <value>jane</value>
 </entry>
</struct>
```

Like entries, structs have an optional name. They provide a means of structuring data items. Structs may contain both named and unnamed entries and structs. A struct containing only named entries can be thought of as a record, a struct containing only unnamed entries is a list. The elements contained in a struct are ordered. Structs may not contain character data (only structs and entries.)

Configurations

```
<configuration target="lcfg-engine"
               version="1.0.3">
 <struct name="applications">
  <entry>
   <value>emacs</value>
  </entry>
 </struct>
</configuration>
```

The configuration element is the root of the XML document. It serves to contain all the other elements. The target attribute indicates the deployment engine for which this configuration is intended, and the version attribute a minimum version number for that engine. This is largely to provide sanity checking (you wouldn't want to accidentally hand a configuration to an old version of the tool and have it trash a machine in confusion.)

Avoiding the Phantom Lexicon

This section has presented one possible approach to defining a "low level" configuration description document format. We have sidestepped the issue of a standard lexicon (that is, defining a standard set of keys whose values should be similarly interpreted by all compliant deployment engines.) We argue that such a standard lexicon could not be meaningfully defined in the present environment, where there is so little commonality between existing tools.

Our purpose is to enable progress towards greater sophistication in configuration management by providing a standard method of interfacing with low level deployment engines. Whilst a standard lexicon would both allow sites using different deployment engines to share configuration data, and allow them to migrate easily between engines, we do not believe there is enough understanding of what the appropriate lexicon would be to define it at present. The most pressing need in this area is to begin to decompose the problem. When the issues involved are better understood, it will be possible to standardise a lexicon. If we cannot start moving forward, we will never reach that point.

The choice of the "level" at which to specify this interface is crucial; if the level is too low (it contains too little structuring information), then it will not be possible for common tools to perform meaningful operations on the configuration. If the level is too high, then the mismatch between the common format and the assumptions of existing tools will be too great, and no meaningful translation will be possible. This proposal is based on a study of the existing tools in the previous section (and others), and we believe that an interface at the level presented is the minimum required to support useful common operations (see the next section), while providing a large degree of architectural freedom for the tools themselves.

Managing Configuration Data

It might seem that the interface set out earlier is too limited to support sophisticated management techniques. In this section, we will attempt to show that a wide range of transformations can be applied to structured configuration data, without needing to have the management tool "understand" that data. There is an analogy here with the use of simple UNIX command line tools, applied in a filter chain, to perform more complex tasks: tools such as sort, grep, m4, and even awk can be combined to perform powerful transformations on text files, even though the programs themselves have very little knowledge of the text file structure apart for a simple division into fields and records.

Classing

The most widespread operation available in configuration tools to manage configuration data is classing: groups of machines are assigned to particular classes, with configuration data being associated with each class. Alternatively, this can be seen as establishing predicates about machines, and associating with each predicate a set of configuration information.

A simple example might involve a "webserver" class, associated with configuration data that specifies webserver specific packages and configuration options. In order to deploy a new webserver, it is only necessary to place it in the "webserver" class, and all the necessary configuration files and packages will be deployed.

Basic classing gives the administrator an enormous increase in the manageability of their network. By separating out roles into classes, it becomes possible for an administrator to create complex combination configurations very quickly. Complex updates can also be rapidly deployed, as changes can be made to class definitions rather than to individual machines: if a bug is found in xinet on Fedora Core 3, it suffices to update the class definition for "Fedora Core 3 machine," which all FC3 machines inherit. There is no possibility of missing, or overlooking a machine, nor is it necessary to modify each machine by hand.

Advanced Classing Operations

Whilst almost all configuration tools provide their own mechanism for managing simple classes of objects, this feature still remains to be fully explored. The simple scenarios described above are both powerful and useful, but as the use of automation matures, administrators find there are unanswered questions in this approach. Consider a "highly-secured" class, and a "web-server" class, each managed by different teams. These classes have different objectives, and

overlapping domains. It is entirely possible specifications from one class will conflict with another. To our knowledge, no current configuration tool presents a satisfactory answer to the question of how conflicts can be resolved.

Current suggestions for progress in this area include:

- **Prioritisation** A machine might belong to different classes with different priorities, enabling the data specified by one class to be outweighed by another. Or classes might set values with different priorities, so allowing them to be safely overridden elsewhere. This would avoid the order-dependent (or even oscillatory) decision process displayed by current tools.
- **Constraints** Rather than specifying definite values for properties, it might be possible to specify ranges of values. This would enable a tool to mediate between different groups automatically. Consider the web-server team specifying that either port 80 or 8080 be used, and the security team that only ports over 1000 be used. These requirements are not conflicting, but had the webserver team simply said port 80, there would be no way for the tool to mediate.

Aggregation

It is common for some aspect of the configuration of a system to be derived from the configurations of other machines. This might be a client being configured with respect to its servers, or a server from its clients. Specifying this information manually is error prone and time consuming. By introducing automation, we both save ourselves time and increase our confidence in the correctness of our network's configuration. Typical examples include:

- A firewall host may want to open holes for services marked as public. Specifying this in the configuration of the hosts, rather than in the configuration of the firewall, means that when the host's service is decommissioned, so is the firewall hole (automatically).
- A DHCP server might want to allocate fixed addresses to individual nodes. Again, the fixed address for a host is most naturally stored with the host itself. This means the server must aggregate those values into its configuration. This can be done systematically by a management tool.

The only tool we are aware of that provides explicit aggregation operations is LCFG, although Quattor and others have the ability to derive parts of a host's configuration from a database. Future research in this area is likely to focus upon the ways in which aggregation can be provided without a central configuration server – it has been suggested that one way to achieve this would be through the use of peer-to-peer technologies.

Again we note that, given just a structured tree of data, it is possible to implement a generic aggregation function, for example by converting values across a group of files into a list of values in the target file.

Sequencing and Planning

Configurations cannot be changed atomically, as there are frequently multiple dependents upon a single value. For example, it may be necessary to both bring up a new server and redirect all clients to point at it, before the old server can be taken down. Thus far we have avoided talking about how configuration changes can be sequenced. To the best of our knowledge, none of the available tools tackle this question.

One possible approach, as yet untested, is to allow the user to specify a set of invariants that must remain true during the deployment of any configuration. These can be used to break down a deployment into multiple steps, each of which maintains the invariants. One example of an invariant might be "A client must not be configured to point at a non-existent server."

Although there can be significant sophistication required to maintain an arbitrary invariant, simple invariants of the kind given above can be shown to generate safe 3-step transitions between configurations. Of course, some integration with monitoring, or user-feedback, would be needed in a real tool to step through the intermediate configurations. It does little good to create several intermediate steps then deploy them all at once!

The important point here is that the specification of invariants can be done in a general way (all that is required is to be able to establish requirements on pairs of values – to the tool it is unimportant what those values represent.) This is an area in which even very simple tools can generate large advances in the state of the art.

Delegation and Authentication

A large installation is not managed by a single person. Often there will be several teams working together, each tasked with maintaining different aspects of the network's configuration. We discussed the possibility of using a configuration management system to perform some automatic mediation between teams. Here we are concerned with the security issues involved in delegating configuration aspects to people outside of the central teams.

Configuration management systems at present have a boolean notion of authorisation: a user is either authorised to perform any action whatsoever on any machine, or they are not authorised to perform any. Although we might feel that there are many people involved in the management of a system who should have some limited control of some aspects of a machine, current tools do not enable us to act upon that idea.

The simplest example is allowing users to control their own machines to some extent. It seems that many sites allow users a choice either of complete

freedom and responsibility, or of accepting a completely prescribed system. Machines may have gigabytes of unnecessary software installed on them because users are unable to choose which software is most applicable to them.

The solution to these problems is to introduce a notion of limited delegation, or authorisation, into our tools. For example a constraint-based system could set hard boundaries on what a user can do to their machine, while still providing much more flexibility than is currently available. Nor would it be necessary for users to understand the tools if unprivileged user-space helpers were available to guide them through the options.

Again, we believe that authorisation and limited delegation are features that could be explored without a known underlying lexicon to work to. Restrictions and security information can be identified with areas of the configuration data tree, and are no less securely or usefully enforced for not being understood by the tool that does so.

An Example

The following example demonstrates how some of the above features might be used to implement the comparatively high-level configuration requirement:

> Configure two DHCP servers on every Ethernet segment. The DHCP servers should provide fixed IP addresses for all the other machines on the segment.

Conventionally, this policy might typically be implemented using a partially automated approach such as the following:

- Two appropriate machines on each ethernet segment might be identified manually.
- The lists of machines on each segment, together with their corresponding IP and MAC addresses may be extracted from some database using an ad-hoc script and massaged into the right form for the configuration files.
- The configuration files would be distributed to the appropriate machines, and the DHCP daemons started on them, possibly with the aid of some tool such as cfengine or LCFG.

Of course, there are many others ways of doing this, but this is certainly typical, and this particular example provides a simple illustration of several important configuration manipulations.

Note that most existing "configuration tools" would be able to handle the last operation automatically, but probably not the first. This illustrates the different between *configuration deployment* and *configuration management*. The manual operations involved in the above configuration management process make it unsuitable for an autonomic environment; if a DHCP servers fails, a new one must be selected manually. The disconnect between the independently created IP database, and the actual deployed machines is also a potential source of errors (if we decommission a machine, can we guarantee that someone will remember to remove the MAC address from the database?).

A configuration management tool must accept the high-level statement of the requirement, and translate it automatically into a form suitable for the deployment engine. For example, this might involve the following process:

- A monitoring system would provide a list of active machines on each ethernet segment.
- A constraint process would be used to select two (active) candidate machines on each segment.
- A classing mechanism would allocate a DHCP-server class to each of these machines. This class would define the appropriate parameters to configure and start the DHCP service.
- An aggregation mechanism would collect the IP/MAC mappings from the individual node configurations and make them available as part of the server configuration.

The result of this process would be a complete, unambiguous configuration specification for each machine which could then be passed to the deployment engine. This solves the two problems of the manual approach noted above; replacements will automatically be configured for failed servers, and machines which are decommissioned will automatically be removed from the DHCP configuration files.

The result of the configuration management process is clearly expressible using the simple document interface outlined earlier. Clients would be configured to run DHCP client software (with no special parameters), and servers would be configured to run DHCP servers (with a specified list of IP/MAC address pairs). The deployment engine would need to translate these simple requirements from the XML description into the appropriate configuration files and daemon operations – a conceptually simple process, but one which may involve a considerable amount of code to handle the details.

It is important to note that most of the above configuration management operations themselves are quite generic; there are many other applications of the operations such as "classing" and "aggregation" – they are not specific to the DHCP example, and can be implemented as generic operations on the simple XML structures proposed earlier.

Conclusions

Advances in the sophistication of modern configuration tools have been hampered by the inability to decompose the problem into subtasks with well defined interfaces. If everyone who wanted to create a programming language chose their character set differently,

wrote their own editor, then created the entire toolchain, we would not see the sophistication and flexibility that we do today.

This paper proposes a simple level of interoperation which should allow independent development of higher-level configuration management tools, without the need to reinvent the wheel at each step. Such tools would benefit from the full range of available deployment engines, and separate out the maintenance burden onto more than one team.

We are sufficiently realistic to recognise that many tools, developed for internal use, will not be modified simply to meet the requirements of some external standard unless there is a significant practical benefit. We do hope, however, that the principles outlined herein will at least influence the development of new configuration tools, and promote a higher degree of interoperability in the future.

We would like to invite comment and discussion of the issues raised here on the lssconf mailing list[1].

Acknowledgements

Many people have been involved in the development of the ideas presented in this paper, and contributed their time to explain tools, and principles. In particular, the authors are grateful to Narayan Desai (Argonne National Laboratories), Luke Kanies (Reductive), Kent Skaar (BladeLogic), John Sechrest (Alpha Omega Computer Systems), Andrew Hume (AT&T Research), and all the participants of the LISA and Edinburgh [4] Configuration workshops.

This work has been partly funded by a grant from the Joint Information System Committee (JISC).[2]

Author Biographies

Paul Anderson is a Principal Computing Officer with the School of Informatics at Edinburgh University, where he divides his time between research projects in System Configuration and practical management of the School's computing infrastructure. He can be reached at dcspaul@inf.ed.ac.uk .

Edmund Smith worked as a researcher in system configuration on the OGSAConfig project at the University of Edinburgh, which finished in 2004. He is currently studying for a Ph.D. in Psychology at the University of Stirling. He can be reached at esmith4@inf.ed.ac.uk .

References

[1] The GGF CDDLM working group, https://forge.gridforum.org/projects/cddlm-wg .

[2] LCFG, http://www.lcfg.org .

[3] Quattor, http://www.quattor.org .

[4] *JISC-sponsored workshop on representations of configuration data*, http://homepages.inf.ed.ac.uk/group/lssconf , April, 2005.

[5] Anderson, Paul, "Towards a high-level machine configuration system," *Proceedings of the 8th Large Installations Systems Administration (LISA) Conference*, pp. 19-26, USENIX, Berkeley, CA, http://www.lcfg.org/doc/LISA8_Paper.pdf, 1994.

[6] Anderson, Paul, George Beckett, Kostas Kavoussanakis, Guillaume Mecheneau, and Peter Toft, *Technologies for large-scale configuration management,* Technical report, The GridWeaver Project, http://www.gridweaver.org/WP1/report1.pdf, December, 2002.

[7] Anderson, Paul and Alastair Scobie, "Large scale Linux configuration with LCFG," *Proceedings of the Atlanta Linux Showcase*, pp. 363-372, USENIX, Berkeley, CA, http://www.lcfg.org/doc/ALS2000.pdf, 2000.

[8] Burgess, Mark, "Cfengine: a site configuration engine," *USENIX Computing Systems*, Vol. 8, Num. 3, http://www.iu.hio.no/mark/papers/paper1.pdf, 1995.

[9] Desai, Narayan, Andrew Lusk, Rick Bradshaw, and Remy Evard, *BCFG: A configuration management tool for heterogeneous environments*, Technical report, Argonne National Laboratory, 2003.

[10] Goldsack, Patrick, *Smartfrog: Configuration, ignition and management of distributed applications,* Technical report, HP Research Labs.

[11] Holgate, Matt and Will Partain, "The Arusha project: A framework for collaborative systems administration," *Proceedings of the 15th Large Installations Systems Administration (LISA) Conference*, USENIX, Berkeley, CA, http://www.usenix.org/events/lisa2001/tech/full_papers/holgate/holgate.pdf, 2001.

[12] LeFebvre, William and David Snyder, "Auto-configuration by file construction: Configuration management with Newfig," *Proceedings of the 18th Large Installations Systems Administration (LISA) Conference*, pp. 93-104, USENIX, Berkeley, CA, http://www.usenix.org/publications/library/proceedings/lisa04/tech/lefebvre.html , 2005.

[1] http://homepages.inf.ed.ac.uk/group/lssconf/

[2] http://www.jisc.ac.uk/ .

A Case Study in Configuration Management Tool Deployment

Narayan Desai, Rick Bradshaw, Scott Matott, Sandra Bittner, Susan Coghlan, Rémy Evard, Cory Lueninghoener, Ti Leggett, John-Paul Navarro, Gene Rackow, Craig Stacey, and Tisha Stacey
– Mathematics and Computer Science Division, Argonne National Laboratory

ABSTRACT

While configuration management systems are generally regarded as useful, their deployment process is not well understood or documented. In this paper, we present a case study in configuration management tool deployment. We describe the motivating factors and both the technical considerations and the social issues involved in this process. Our discussion includes an analysis of the overall effect on the system management model and the tasks performed by administrators.

Introduction

System administration is, at its heart, the profession of helping people to use computers. System administrators are domain experts who provide impedance matching between users' desires and computers. The expertise of system administrators is manifested in their choices of computer hardware and software and of system configuration. Environments are constantly changing; the need for timely software updates and frequent configuration changes has never been greater. Configuration management tools provide different levels of automation and representational models, but all have the same basic goal: to help in the system configuration process.

Nevertheless, adoption of configuration management systems has lagged substantially behind tool development. The explanation rests largely with the up-front cost of building an adequate representation of an environment, but the problem can also be traced to the requirement that administrators change their administration methods. This change makes the configuration management adoption process quite costly and time consuming.

The Mathematics and Computer Science Division of Argonne National Laboratory consists of nearly 200 researchers, programmers, students, and visitors. The division is home to several hundred workstations, three large clusters, and other high-performance computing resources. One group maintains this diverse set of resources. Recognizing the need for more efficient management mechanisms, members of the system staff have contributed to a number of system management research efforts [6, 4, 7].

In the summer of 2004, the group decided to deploy Bcfg2 [3], a configuration management tool developed in house. As of May 2005, much of the deployment has been completed. Bcfg2 manages the general infrastructure and two clusters and is being deployed on another cluster and on an IBM Blue Gene/L system. The general infrastructure is extremely complex and involves the largest number of administrators, hence; we focus on this particular deployment.

This paper documents our experiences and lessons learned in adopting a configuration management tool. We discuss the goals that prompted this adoption and describe the deployment process in terms of both technical and – more important – social issues. The adoption of Bcfg2 has dramatically changed the procedures and model used by system administrators in the division. We discuss these outcomes in detail.

One cannot talk about configuration management without including technical aspects that are likely tool-specific. As the same time, we believe that many of the issues we faced and benefits we reaped in deploying Bcfg2 are intrinsic to the adoption of any configuration management system. Where possible, we avoid the discussion of Bcfg2-specific topics.

Since configuration management has been recognized as a problem for quite some time [4, 9], numerous tools have been written to address this task. Many of the tools, in particular LCFG [1] and Quattor [8], can cause large changes in management methods; our discussion of Bcfg2 [3] is especially applicable to such tools. Other tools, such as SystemImager [5] and CFEngine [2], have a model that is not substantially different from manual system administration; for these tools our discussion of deployment issues is less applicable.

Architecture

In this section, we describe the overall goals that motivated our adoption of Bcfg2. We also briefly describe Bcfg2, in order to provide an understanding of the outcomes we achieved.

Goals

The bulk of the benefits we hoped to achieve with better configuration management were efficiency related. We were spending an inordinate amount of

time performing software upgrades and applying security patches. In partial response to this problem, we found ourselves faced with the deployment of a new base operating system, Debian Sarge. This occasion provided an ideal opportunity to add a new configuration management system into the environment.

We wanted a centralized location where all configuration information for all clients could be stored and coordinated. This requirement raised a variety of expressivity issues. We wanted all aspects of the configuration specification to be as terse as possible. That is, we wanted to express all required configurations in a way that minimized redundancy in the specification. Similarly, we wanted configuration changes to be made in as simple a way as possible. For example, adding a software upgrade to all systems or uniformly changing the contents of a configuration file should be trivial.

We also wanted a high-level interface into the configuration management that allowed configuration specification based on machine role. For example, it should be easy to ask for another instance of a given role, like a web server.

Another important goal was to eliminate configuration state local to clients. That is, all machines should match their configuration specification. Once no local configuration exists, no local configuration data will be lost in the event of a machine rebuild. Adherence to this goal enables machines to be trivially rebuilt. Furthermore, once the configuration specification contains all configuration directives needed to produce a goal, this goal can be replicated.

Finally, we wanted good practices encoded in the configuration management tool. The tool should perform operations in the safest manner possible. It should also make every attempt to ensure that new configuration specifications are activated.

Bcfg2 Overview

While the Bcfg2 architecture is not the main focus of this paper, several details are required to understand our deployment. Bcfg2 has a client-server architecture. The server is responsible for building configurations for clients, based on a global configuration specification. This specification contains high-level directives for clients, referred to as the metadata, and a set of configuration rules that can be used with the metadata to produce literal client configurations. That is, they contain information that needs no further processing for client use. These configurations are also assumed to be comprehensive; they contain information for all aspects of client configuration. For example, a software package or service that is active on a client will be included in its configuration. Any installed configuration entities not listed in the configuration are flagged as "extra." The Bcfg2 client uses heuristics to discover this "extra" configuration.

The Bcfg2 client connects to the server and downloads its configuration. It then inventories the local system and compares this inventory to the configuration specification. Any discrepancies found in this process are flagged for later correction. Next, the Bcfg2 client runs its heuristics to find extra configuration. Anything located in this pass is similarly flagged for later correction. After the detection work is done, the Bcfg2 client rectifies any conditions previously found. This behavior is tunable; dry-run and extra configuration removal modes are available.

Once the client has completed operations, it uploads statistics to the server. The information includes the overall machine state (clean or dirty) and lists of the failing and modified configuration entries. This information is stored on the server and can be used to generate nightly reports about the overall state of the network and its correspondence to the configuration specification.

Deployment

Deploying Bcfg2 in our environment took a substantial amount of time. Our experiences since its deployment, however, have more than justified this investment. In this section we describe the technical and social issues involved in the process. Following this, we discuss various improvements to the administrative process enabled by this deployment.

Technical Deployment

Deploying Bcfg2 took approximately four months of work performed primarily by one person. As is frequently the case with systems like this, the first 90 percent of the work was completed in the first six weeks, while numerous small issues were resolved over the course of the next 10 weeks.

Deployment was initially undertaken as part of a base OS upgrade, moving from Redhat 7.3 to Debian Sarge. We chose this occasion because our previous experience with a mid-stream switch to Bcfg1 had proved problematic. Basically, systems managed in an ad hoc fashion tend to have a lot of configuration inconsistencies. Ensuring that these machines can be cleanly rebuilt by using a configuration management tool can be quite difficult. The introduction of a configuration management tool is possible but must be carefully performed. In contrast, the deployment of new machines can be easily done, as the deployment process is available for examination.

The first goal was to get an automated build system working properly. Our environment has two main types of machines: workstations and servers. We decided to use SystemImager [5] for initial client installations, calling Bcfg2 before initial reboot. This approach ensures that machines come up properly configured and secure upon first reboot. Two profiles for Bcfg2 were created and made selectable from the initial SystemImager boot menu.

Creating the configuration profiles for workstations and servers was not difficult. This process consisted of

specifying all configuration aspects of workstations, including all environment-specific modifications. A workstation configuration had been created for Bcfg1 and was easily migrated to Bcfg2. Had this not been the case, the creation of an initial configuration would have taken a few days. This process consists of recording all important aspects of configuration in the central specification. Aspects can be quickly incorporated into the specification. Our workstation configuration initially consisted of nearly 1,100 configuration aspects. Once this process was completed, we constructed a server configuration as a subset of the workstation configuration, since many of the configuration aspects of these classes are similar.

Once a simple build mechanism was in place, we rebuilt administrator desktops and some test servers using the new image and management system. In one case, the administrator ran with two desktops concurrently for several weeks, in order to find subtle aspects of needed workstation configurations not yet included in the workstation profile. The process paid a big dividend; by the time "normal" users started using workstations based on the new build, few configuration problems remained.

After the new system became full-featured and stable enough, we started to allow users to request machines with it. These early users provided the remainder of input regarding missing configuration from the new workstations. After several of these users had successfully used new-build machines for some time, we began upgrades for the rest of the division.

In this stage the simplified build process was particularly beneficial. Three people performed most of the workstation rebuilds. The initial target was to complete most machine upgrades in four weeks. Desktop rebuilds were a simple process. Each machine has a local scratch disk, which gets cleaned upon rebuild. Users save any needed data stored there. Once this data is saved, rebuilds can occur at any time and take 30 minutes. All user interactions occur in the first three minutes of the process; the rest of the process can complete unattended. In the course of a month, nearly 80 desktop machines were upgraded.

After completing the workstation upgrades, we shifted our focus to the server machines. While the server profile had been used to build new machines, many existing servers remained unmanaged by Bcfg2. As these servers were replaced, we encountered a whole new set of issues relating to tool deployment – issues that were generally related not to tool completeness but rather to the way configuration changes were propagated to machines. For example, certain machines were deemed too important to perform automatic changes.

This realization necessitated a major change in our deployment strategy. Initially, all machines had called Bcfg2 each night, and all required changes were performed. On workstations, this model was good enough; while these machines are important, they don't cause congestive failures when problems occur. For servers, however, we needed to run Bcfg2 in dry-run mode each night and send its state to the relevant administrator. While examining this issue, we also realized that cluster nodes should run Bcfg2 in yet another way: between jobs, in order to prevent interference with user calculations.

These experiences shifted our focus from a tool-based one to an administrator experience-based one. Initially, we were quite concerned about tool correctness and completeness. As Bcfg2 proved itself and bugs were fixed, our confidence in its correctness greatly increased. Also, our workstation configuration proved to be as complicated as any other in our environment, so the configuration specification process for our servers was fairly simple.

Our change in focus amounted to the realization that the tool client-side functionality was not sufficient. The tool must also provide enough information for administrators to make effective decisions as conveniently as possible. Moreover, it must supply configuration state information in a convenient way. From this point onward, nearly all development focused on an information presentation system to provide a sort of scoreboard for the entire network and its configuration state.

Administrative Improvements

The deployment culminated with the implementation of a robust reporting infrastructure. This infrastructure provides periodic information about the current configuration state of all clients, the time of their last contact with Bcfg2, and a list of pending configuration changes. The information allows administrators to observe the logic employed by Bcfg2 during normal operations. This single factor had more impact on administrator trust in Bcfg2 than all others.

The reporting system results in emails, Web pages, or RSS feeds that contain information about either specific hosts or the overall status of all Bcfg2 clients. Administrators can subscribe to reports about particular clients of interest, or all systems, enabling the detection of two common problems: clients not receiving configuration updates and clients unable to perform needed configuration updates. Administrators can also observe the operations taken by Bcfg2 over time.

These reports create a picture of the entire network that gives discussions about configuration a basis in fact. The reports greatly improved our understanding of our systems, their configuration, and its modification patterns. More important, administrators grew to trust Bcfg2, and hence allow the remainder of our network, composed of our most important machines, to be rebuilt and managed by Bcfg2.

Social Issues

While the technical aspects of Bcfg2 deployment were complex, managing the social aspects of the

deployment was far more difficult. This process occurred in an ad hoc way in our group and could have been substantially improved had we initially known the related considerations.

Adoption of any configuration management system is a stressful process. Configuration management tools affect the whole of the system management process. All administrators are required to adopt new processes for achieving the same tasks they already know how to accomplish. Since such changes can directly impact the services system administrators are expected to provide reliably, tensions can run high.

Communication also becomes an issue, because of the variety of perspectives held by different administrators. We found that three main factors motivated most of our disagreements during the deployment process: trust in the tool, a belief in the benefits provided by the tool, and the perception of a complexity increase or decrease caused by the tool.

Trust is certainly the most important of these factors. If a tool remains untrusted by administrators, it will never be substantially used in their environment. The trust-earning process certainly varies from person to person, but we can discuss the issues we observed. In general, as users gain more experience with the configuration management tool, they begin to trust it more. Two aspects of trust are important. The first is that the tool can properly represent any configuration state that the administrators may desire. This problem is largely technical and is solved as the administrator gains experience and familiarity with the tool. The second aspect of trust is harder to earn. Administrators must trust that the tool will perform the specified changes appropriately. This sort of trust is earned only through a long period of experience running the tool and observing the resulting configuration changes. The process can be accelerated, however, through the exposure of tool decision information. If administrators can easily examine the decision process each time they run the tool, then their trust will grow more rapidly.

The second factor that guides the adoption process is the *perception of benefit.* All administrators in our group were already overburdened with tasks, so expecting them to spend time learning something new was difficult. If a management tool didn't present a clear case for rapid improvement in efficiency, learning about it wouldn't be given high priority – and rightly so. Lack of time to experiment with a tool clearly has a detrimental effect on overall trust in the tool. Hence, adoption can be greatly hampered simply because of a lack of information. This factor can be minimized, however, by providing improvements that quickly benefit all administrators. Easy-to-adopt solutions to common problems provide a good incentive to try out a tool.

The third factor in the adoption of configuration management systems is the *perception of complexity.* The complexity increase may be minor; but to users unfamiliar with particular tool, this added complexity will be unattractive because it will lead to decreased efficiency for some tasks. Over time, as the administrators become more familiar with the tool, the added cost of this complexity is reduced. Once the deployment is complete, complexity is compartmentalized, as administrators can focus on their task and ignore others. For example, an administrator responsible for web servers can focus on `apache` configuration without worrying about upgrading `ssh`.

All of these factors are heavily interrelated. Throughout the deployment process, their influence was felt through each of the issues we encountered.

Disagreements

Throughout the process of tool adoption, each administrator in the group internalizes more information about the new tool, building an opinion about the tool's value and potential use cases. Each administrator will trust the tool to a different extent and will have different ideas about the potential benefits to be gained and the complexity costs involved. The following discussion describes the different points of contention that arose in our group. Each of the major issues is described in the abstract, along with the factors that turned out to be important. While we believe that these issues are representative, the list is by no means comprehensive. People like to argue about all sorts of issues.

- Buy-in. Initially, everyone must agree that a given tool should be used. This issue is largely influenced by the trust and comfort levels administrators have with a tool, not to mention its technical correctness. Learning how a tool works is essential, but this process can require substantial amounts of time, especially in large groups.
- Existing investments. A working environment typically has a sizable investment in technical methodologies. Generally, a set of utilities has been developed to automate particular tasks, and a large body of institutional knowledge has evolved around specific tools and methodologies. Moreover, the creation of these processes and tools usually implies an emotional investment. Overcoming these investments without alienating members of the group is difficult – but possible.
- Level of control. Any tool will have a fundamental set of assumptions or functionality that guides its behavior. Even if the tool is reliable, it can be difficult to convince everyone that a change is beneficial – especially if the methods that a proposed new tool uses to implement a given task differ from those historically used. This issue can be overcome only with a large amount of empirical evidence that these methods are equivalent. As when programmers made the initial switch to high-level languages, administrators are accustomed to making complex low-level changes to systems, rather than using high-level specifications of functionality.

Our goal is not to diminish these concerns in any way; in fact, all of these considerations are quite reasonable. In some ways, these concerns illustrate the core essence of system administration. That is, the adoption of a tool that will radically affect every aspect of system maintenance is not a decision that should be taken lightly. Administrators, after all, shoulder the brunt of system failures, misconfigurations, and software configuration problems.

Rather, our intent is to document these concerns. We believe we have gained a deeper understanding of our requirements and also provided a comprehensive vetting process. Moreover, through this documentation, we hope to aid other groups attempting the same sort of transition.

The highest-level problem can be most easily summarized as "buy-in." Once that problem is resolved, the tool deployment will achieve critical mass and no longer serves as a point of contention.

Hindsight

Despite the social issues we confronted, we managed to implement a configuration management system. We attribute our success to three factors.

- Our group was predisposed to recognizing the value of configuration management. This attitude removed an important initial hurdle from the process. Had we simultaneously needed to convince the group of both the need and mechanism for configuration management, the outlook would have been far worse.
- One administrator was involved in both the Bcfg2 development activities and maintenance of the division infrastructure. Without his work as liaison, many arguments wouldn't have carried nearly the weight that they did; adoption could have easily stalled.
- Our group is quite amicable, and not particularly sentimental. These characteristics allowed easier discussion of contentious subjects and the replacement of existing mechanisms and tools.

Even with these factors, considerable perseverance and evangelism were required. The outcome of this process was in doubt throughout much of the deployment process. At many points, administrators did not seem to find the model compelling enough and did not trust the tool. Fortunately, everything worked out well in the end.

Recommendations

While there is a great amount of social variance among groups of system administrators, we can make several recommendations based on our experiences.

- The tool under consideration must have an advocate who is technically respected by the group. His role will be to assess the various tools and select one that seems best suited to the environment. He will also need to convince other members of the group that the chosen tool is the proper one.
- Administrator concerns should be addressed, not ignored. These concerns are generally based on experience and reflect potential technical issues that could occur later. Once all concerns have been addressed, the group will more readily accept the tool into daily operations.
- Tool advocacy will be most compelling when improvements mentioned are useful in the short or medium term. Long-term improvements, while desirable, do not often provide short-term motivation. Hence, long-term benefits secure a position on the "when we have time" list for tool deployment.

Outcomes

In spite of the difficulties described above, this project has succeeded beyond our expectations. We have several new capabilities we could not have predicted even six months ago. All of these capabilities have resulted from three major shifts in architecture. First, we now have a centralized configuration specification and statistics that allow reasoning about the desired (and actual) configuration states of our entire environment. Second, we now have an abstraction barrier between our specifications of machine function and the implementation of that function. This barrier simplifies both parts of the configuration specification and allows easier interactions. Third, several operations have been completely implemented by the configuration management system, thereby reducing the cost in time, and improving the economies of many processes. Each of these improvements contribute to a fundamental alteration of the management model for our environment.

Configuration Specification

Many configuration management tools have a centralized specification that describes the desired state of the network. Where Bcfg2 departs from this model is the addition of detailed statistics about client state. The addition of these statistics to the central specification causes what would be a static set of rules to become a living document, automatically updated by Bcfg2. The desired configuration and all deviations from it are available in a central location.

The existence of a configuration oracle for an entire network alters the administration mind-set in that global notions of state can now be constructed. Many data mining techniques can now be utilized. Reports describing network configuration state, reporting on the frequency and success of client configuration processes can give a thorough picture of unexpected client states before users are affected. Similarly, reports automatically generated from the configuration specification can provide insight into current software revisions, client functionality, and potentially even system interdependence. Reports such as these can prove useful for auditing purposes, the training of

new employees, and high-level descriptions of services provided.

Function Abstraction

The configuration specification used by Bcfg2 splits information into three layers. The first, called the metadata layer, describes function information in terms of clients and classes. For example, the metadata may describe a class of clients that include web server functionality. The second layer, called the repository, ascribes meaning to those descriptions. In the same example, the repository would contain information describing what "being a web server" means. The third layer, implementing client reconfiguration operations, receives configurations from the higher two layers and reports on execution results. While this architecture is Bcfg2 specific, many other complex configuration management tools are structured in a similar way [1].

This layered specification provides an abstraction barrier isolating function assignment from function description. For example, after "being a web server" is defined, one can easily and reliably add new web server instances. Once this operation is possible, programs that generate these changes become possible. The addition of logic into this function determination process introduces dramatic flexibility into the network. It also allows common function shifting operations to be automated.

Repository semantics benefit from this abstraction barrier as well. Several simple context-specific formats can be used to describe implementation behavior. Similarly, scripts can be written to autogenerate these files, if desired. Important functionality, ranging from automated patch integration to service reconfiguration, can be implemented.

The client-side tool receives a literal set of configuration directives from the Bcfg2 server. It compares the current operational state with the desired state and performs a set of state transitions, focusing on safely transitioning into the goal state. The client tool implements a small number of operations that have been well tested. Upon completion, the Bcfg2 client returns a set of statistics about the configuration state of the client and modifications performed. In conjunction, these two features allow administrators to focus on configuration goals, while allowing the client tool to determine a reasonable set of operations to implement these goals. In this way, the client tool isolates the upper-level users from some low-level complexity.

Tool-Based Simplification

Tools are intended to make tasks easier. Using Bcfg2, we found three major tasks that were vastly simplified.

System rebuilds have become trivial. Statistics are used to verify that the configuration specification matches the running state of the machine. After this match has been verified, the machine can be rebuilt at the appropriate time. The previous process consisted of a lot of manual verification; a second system would typically be used to verify functionality and swapped in after everything worked.

The new machine build process has also been greatly accelerated and simplified. Several stock profiles are available; the user is presented with a menu at the beginning of the build process. No other setup is required. Hence, non-root users are now able to build machines quite easily.

The class-based system allows new profiles to be easily created by composing existing function groups. Hence, users can create new profiles whenever needed, without shoehorning functions into the same profile. In this fashion, the configuration specification becomes more concise and representative of the actual running configuration.

Each of these tasks has become easier with the addition of a configuration management system. While these tasks could previously be performed, more expertise and privileges were frequently required. The net impact of these changes is to empower users to be able to perform tasks that were not previously possible.

Overall Efficiency Gains

All of these factors contributed to an impressive increase in efficiency. We estimate that before conversion three FTEs of time were spent on the maintenance of our workstation and server environment. These administrators performed a variety of tasks, ranging from security updates to new software installation, and user-requested reconfigurations. After conversion, between one-third and one-half of an FTE is consumed by these activities.

The time freed by these improvements is now available for a variety of activities. Large-scale infrastructure improvement projects are under way. More time for interactions with users has resulted in more satisfying services for users and more accurate assessments of user needs.

Basic administration tasks remain split across several administrators. The use of configuration management has also reduced the cost of task distribution. The existence of a central location for configuration specification imposes a set of expectations that allow administrators to find one another's work and synchronize when needed. Most simply, all administrators are kept on the same page.

Conclusions and Future Work

Our main goal in this paper is to document the process and results of deploying a configuration management tool. While the process is difficult, the outcomes are worthwhile. Indeed, the outcomes we experienced have more than justified the effort involved. While some of the difficulties faced were certainly peculiar to our group, we feel that the ones documented here are indicative of the fundamental issues in a change of this magnitude.

While our discussion may suggest that this task is too difficult for many groups to consider, we strongly believe that configuration management is an important technology to deploy in nearly any environment. We hope that our discussion of difficulties will not dissuade other groups but, rather, will be used to navigate an admittedly difficult process.

Many administrative tasks have been vastly simplified, and much useful data can be mined from the configuration specification and statistics. The availability of this data has enabled a higher level of reporting and comprehension of our environment than was previously attainable.

While many improvements have already been realized, further substantial efficiency gains can be achieved. Certainly, Bcfg2 could be improved. Several technical improvements relating to information representation require attention. Also, an interface to force client reconfigurations would be useful. The user interface will also continue to improve with more experience.

As a group, we expect administrative model changes to continue, although with a less disruptive effect. Many processes remain that could be automated. Also, a service that allows reconfiguration delegation would be useful, allowing users to update aspects of configuration as appropriate. Moreover, we hope that administration streamlining will continue.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U. S. Department of Energy, under Contract W-31-109-ENG-38.

Author Biographies

Narayan Desai has worked for 5 years as a systems administrator and developer in the Mathematics and Computer Science Division of Argonne National Laboratory. His primary focus is system software issues, particularly pertaining to system management and parallel systems. He can be reached at desai@mcs.anl.gov.

Rick Bradshaw holds a BS in Computer Science from Edinboro University. He has been a member of the MCS Systems team since 2001, where he aids in maintaining HPC resources, experimental computing resources, and general UNIX infrastructure. He can be reached at bradshaw@mcs.anl.gov.

Scott Matott is a network engineer for UBS AG. Previously, he worked as a network engineer at Argonne National Laboratory and the University of Chicago. He can be reached at scott.matott@gmail.com.

Cory Lueninghoener is a smug and condescending HPC Systems Administrator at Argonne National Laboratory working on the Teragrid cluster. Prior to this current life, he also spent lives as both an undergraduate and graduate student at the University of Nebraska-Lincoln working with the Research Computing Facility. He can be reached at lueningh@mcs.anl.gov.

Ti Leggett is a systems administrator for the Futures Laboratory of the Mathematics and Computer Science Division at Argonne National Laboratory. He also has a joint appointment with the Computation Institute at the University of Chicago and can be reached at leggett@mcs.anl.gov.

Gene Rackow has been the curmudgeon of the systems group of the Mathematics and Computer Science Division since before there was a systems group. He has been instrumental in the operation of many generations of HPC platforms used by the division over the last 25 years. More recently, his attentions have officially turned to security issues, where he will be taking more of a labwide role. He can be reached at rackow@mcs.anl.gov.

Susan Coghlan has been managing HPC systems and HPC system administrators for several years in the MCS Division of Argonne National Laboratory. Prior to that, she helped to administrate ASCI Blue Mountain, a 6144 processor supercomputer at Los Alamos National Laboratory. When not fiddling with some of the world's largest computers, she does so with other Irish traditional musicians.

Rémy Evard is the CIO of Argonne National Laboratory. Prior to this he performed several roles in the Mathematics and Computer Science Division related to managing systems and administrators. His research interests include configuration management and high-performance computing. He can be reached at evard@anl.gov.

John-Paul Navarro has been a high-performance system administrator in the Mathematics and Computer Science Division at Argonne National Laboratory since 1997. In that time, he has operated a variety of HPC resources, including IBM SPs and several clusters. His current research interests include distributed high-performance computing, storage systems, resource management and scheduling, and relational databases. He can be reached at navarro@mcs.anl.gov.

Craig Stacey has worked as a Systems Administrator and Information Technology Manager for 8 years in the Mathematics and Computer Science Division at Argonne National Laboratory. His research interests focus primarily on the intersection of robots, monkeys and pants. His email address is stace@mcs.anl.gov.

Tisha Stacey felt most comfortable describing herself with this haiku:

I'm Tisha Stacey.
I work as a sysadmin.
Please don't e-mail me.

Sandra Bittner joined the Systems Group at Argonne National Laboratory in 1997. She was lead

on the installation of a 128-processor SGI Onyx 2 with 12 graphics pipes and a core member of the NSF funded TeraGrid project, which is building the world's largest, fastest distributed infrastructure for open science research. She is an active member of ACM, IEEE, NSPE, and Usenix/Sage. She holds a bachelor's degree in computer engineering from the University of Illinois at Chicago.

References

[1] Anderson, Paul and Alastair Scobie, "Large scale Linux configuration with LCFG," *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, pp. 363-372, Atlanta, Georgia, USA, October 10-14, 2000.

[2] Burgess, Mark, "Cfengine: A site configuration engine," *USENIX Computing Systems*, Vol. 8, Num. 3, pp. 309-402, 1995.

[3] Desai, N., R. Bradshaw, R. Evard, and A. Lusk, "Bcfg: A configuration management tool for heterogeneous environments," *Proceedings of the 5th IEEE International Conference on Cluster Computing (CLUSTER03)*, pp. 500-503, IEEE Computer Society, 2003.

[4] Evard, Rémy, "An analysis of UNIX system configuration," *Proceedings of the Eleventh Systems Administration Conference (LISA XI)*, pp. 179-194, USENIX, Berkeley, October 26-31, 1997.

[5] Finley, Brian Elliot, "VA SystemImager," *Proceedings of the 4th Annual Linux Showcase and Conference*, pp. 181-186, Atlanta, Georgia, October 10-14, 2000.

[6] Gomberg, Michail, Rémy Evard, and Craig Stacey, "A comparison of large-scale software installation methods on NT and UNIX," *Proceedings of the Large Installation System Administration of Windows NT Conference*, pp. 37-47, USENIX, Berkeley, CA, August 5-8, 1998.

[7] Navarro, J. P., R. Evard, D. Nurmi, and N. Desai, "Scalable cluster administration – Chiba City I approach and lessons learned," *Proceedings of the 4th IEEE International Conference on Cluster Computing (CLUSTER02)*, pp. 215-221, IEEE Computer Society, 2002.

[8] Poznánski, Piotr, German Cancio Meliá, Rafael García Leiva, and Lionel Cons, "Quattor – a framework for managing grid-enabled large-scale computing fabrics," *Proceedings of the Kracow Grid Workshop '04*, Kracow, December, 2004.

[9] Traugott, Steve and Joel Huddleston, "Bootstrapping an infrastructure," *Proceedings of the Twelfth Systems Administration Conference (LISA XII)*, pp. 181-196, USENIX, Berkeley, CA, USA, 1998.

Reducing Downtime Due to System Maintenance and Upgrades

Shaya Potter and Jason Nieh – Columbia University

ABSTRACT

Patching, upgrading, and maintaining operating system software is a growing management complexity problem that can result in unacceptable system downtime. We introduce AutoPod, a system that enables unscheduled operating system updates while preserving application service availability. AutoPod provides a group of processes and associated users with an isolated machine-independent virtualized environment that is decoupled from the underlying operating system instance. This virtualized environment is integrated with a novel checkpoint-restart mechanism which allows processes to be suspended, resumed, and migrated across operating system kernel versions with different security and maintenance patches.

AutoPod incorporates a system status service to determine when operating system patches need to be applied to the current host, then automatically migrates application services to another host to preserve their availability while the current host is updated and rebooted. We have implemented AutoPod on Linux without requiring any application or operating system kernel changes. Our measurements on real world desktop and server applications demonstrate that AutoPod imposes little overhead and provides sub-second suspend and resume times that can be an order of magnitude faster than starting applications after a system reboot. AutoPod enables systems to autonomically stay updated with relevant maintenance and security patches, while ensuring no loss of data and minimizing service disruption.

Introduction

As computers become more ubiquitous in large corporate, government, and academic organizations, the total cost of owning and maintaining them is becoming unmanageable. Computers are increasingly networked, which only complicates the management problem, given the myriad of viruses and other attacks commonplace in today's networks. Security problems can wreak havoc on an organization's computing infrastructure. To prevent this, software vendors frequently release patches that can be applied to address security and maintenance issues that have been discovered. This creates a management nightmare for administrators who take care of large sets of machines. For these patches to be effective, they need to be applied to the machines. It is not uncommon for systems to continue running unpatched software long after a security exploit has become well-known [22]. This is especially true of the growing number of server appliances intended for very low-maintenance operation by less skilled users. Furthermore, by reverse engineering security patches, exploits are being released as soon as a month after the fix is released, whereas just a couple of years ago, such exploits took closer to a year to create [12].

Even when software updates are applied to address security and maintenance issues, they commonly result in system services being unavailable. Patching an operating system can result in the entire system having to be down for some period of time. If a system administrator chooses to fix an operating system security problem immediately, he risks upsetting his users because of loss of data. Therefore, a system administrator must schedule downtime in advance and in cooperation with users, leaving the computer vulnerable until repaired. If the operating system is patched successfully, the system downtime may be limited to just a few minutes during the reboot. Even then, users are forced to incur additional inconvenience and delays in starting applications again and attempting to restore their sessions to the state they were in before being shutdown. If the patch is not successful, downtime can extend for many hours while the problem is diagnosed and a solution is found. Downtime due to security and maintenance problems is not only inconvenient but costly as well.

We present AutoPod, a system that provides an easy-to-use autonomic infrastructure [11] for operating system self-maintenance. AutoPod uniquely enables unscheduled operating system updates of commodity operating systems while preserving application service availability during system maintenance. AutoPod provides its functionality without modifying, recompiling, or relinking applications or operating system kernels. This is accomplished by combining three key mechanisms: a lightweight virtual machine isolation abstraction that can be used at the granularity of individual applications, a checkpoint-restart mechanism that operates across operating system versions with different security and maintenance patches, and

an autonomic system status service that monitors the system for system faults as well as security updates.

AutoPod provides a lightweight virtual machine abstraction called a POD (PrOcess Domain) that encapsulates a group of processes and associated users in an isolated machine-independent virtualized environment that is decoupled from the underlying operating system instance. A pod mirrors the underlying operating system environment but isolates processes from the system by using host-independent virtual identifiers for operating system resources. Pod isolation not only protects the underlying system from compromised applications, but is crucial for enabling applications to migrate across operating system instances. Unlike hardware virtualization approaches that require running multiple operating system instances [4, 29, 30], pods provide virtual application execution environments within a single operating system instance. By operating within a single operating system instance, pods can support finer granularity isolation and can be administered using standard operating system utilities without sacrificing system manageability. Furthermore, since it does not run an operating system instance, a pod prevents potentially malicious code from making use of an entire set of operating system resources.

AutoPod combines its pod virtualization with a novel checkpoint-restart mechanism that uniquely decouples processes from dependencies on the underlying system and maintains process state semantics to enable processes to be migrated across different machines. The checkpoint-restart mechanism introduces a platform-independent intermediate format for saving the state associated with processes and AutoPod virtualization. AutoPod combines this format with the use of higher-level functions for saving and restoring process state to provide a high degree of portability for process migration across different operating system versions that was not possible with previous approaches. In particular, the checkpoint-restart mechanism relies on the same kind of operating system semantics that ensure that applications can function correctly across operating system versions with different security and maintenance patches.

AutoPod combines the pod virtual machine with an autonomous system status service. The service monitors the system for system faults as well as security updates. When the service detects new security updates, it is able to download and install them automatically. If the update requires a reboot, the service uses the pod's checkpoint-restart capability to save the pod's state, reboot the machine into the newly fixed environment, and restart the processes within the pod without causing any data loss. This provides fast recovery from system downtime even when other machines are not available to run application services. Alternatively, if another machine is available, the pod can be migrated to the new machine while the original machine is maintained and rebooted, further minimizing application service downtime. This enables security patches to be applied to operating systems in a timely manner with minimal impact on the availability of application services. Once the original machine has been updated, applications can be returned and can continue to execute even though the underlying operating system has changed. Similarly, if the service detects an imminent system fault, AutoPod can checkpoint the processes, migrate, and restart them on a new machine before the fault can cause the processes' execution to fail.

We have implemented AutoPod in a prototype system as a loadable Linux kernel module. We have used this prototype to securely isolate and migrate a wide range of unmodified legacy and network applications. We measure the performance and demonstrate the utility of AutoPod across multiple systems running different Linux 2.4 kernel versions using three real-world application scenarios, including a full KDE desktop environment with a suite of desktop applications, an Apache/MySQL web server and database server environment, and a Exim/Procmail e-mail processing environment. Our performance results show that AutoPod can provide secure isolation and migration functionality on real world applications with low overhead.

This paper describes how AutoPod can enable operating system self-maintenance by suspending, resuming, and migrating applications across operating system kernel changes to facilitate kernel maintenance and security updates with minimal application downtime. Subsequent sections describe the AutoPod virtualization abstractions, present the virtualization architecture to support the AutoPod model, discuss the AutoPod checkpoint-restart mechanisms used to facilitate migration across operating system kernels that may differ in maintenance and security updates, provide a brief overview of the AutoPod system status service, provide a security analysis of the AutoPod system as well as examples of how to use AutoPod, and present experimental results evaluating the overhead associated with AutoPod virtualization and quantifying the performance benefits of AutoPod migration versus a traditional maintenance approach for several application scenarios. We discuss related work before some concluding remarks.

AutoPod Model

The AutoPod model is based on a virtual machine abstraction called a pod. Pods were previously introduced in Zap [16] to support migration assuming the same operating system version is used for all systems. AutoPod extends this work to enable pods to provide a complete secure virtual machine abstraction in addition to heterogeneous migration functionality. A pod looks just like a regular machine and provides the same application interface as the underlying operating system. Pods can be used to run

any application, privileged or otherwise, without modifying, recompiling, or relinking applications. This is essential for both ease-of-use and protection of the underlying system, since applications not executing in a pod offer an opportunity to attack the system. Processes within a pod can make use of all available operating system services, just like processes executing in a traditional operating system environment. Unlike a traditional operating system, the pod abstraction provides a self-contained unit that can be isolated from the system, checkpointed to secondary storage, migrated to another machine, and transparently restarted.

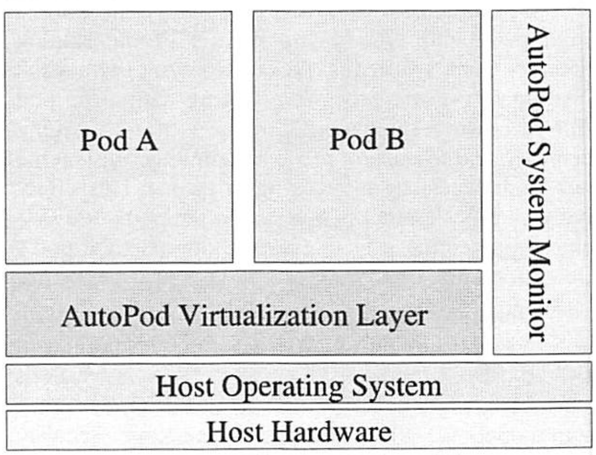


Figure 1: The AutoPod model.

AutoPod enables server consolidation by allowing multiple pods to be in use on a single machine, while enabling automatic machine status monitoring as shown in Figure 1. Since each pod provides a complete secure virtual machine abstraction, they are able to run any server application that would run on a regular machine. By consolidating multiple machines into distinct pods running on a single server, one improves manageability by limiting the number of physical hardware and the number of operating system instances an administrator has to manage. Similarly, when kernel security holes are discovered, server consolidation improves manageability by minimizing the amount of machines that need to be upgraded and rebooted. The AutoPod system monitor further improves manageability by constantly monitoring the host system for stability and security problems.

Since a pod does not run an operating system instance, it provides a virtualized machine environment by providing a host-independent virtualized view of the underlying host operating system. This is done by providing each pod with its own virtual private namespace. All operating system resources are only accessible to processes within a pod through the pod's virtual private namespace.

A pod namespace is private in that only processes within the pod can see the namespace. It is private in that it masks out resources that are not contained within the pod. Processes inside a pod appear to one another as normal processes that can communicate using traditional Inter-Process Communication (IPC) mechanisms. Other processes outside a pod do not appear in the namespace and are therefore not able to interact with processes inside a pod using IPC mechanisms such as shared memory or signals. Instead, processes outside the pod can only interact with processes inside the pod using network communication and shared files that are normally used to support process communication across machines.

A pod namespace is virtual in that all operating system resources including processes, user information, files, and devices are accessed through virtual identifiers within a pod. These virtual identifiers are distinct from host-dependent resource identifiers used by the operating system. The pod virtual namespace provides a host-independent view of the system by using virtual identifiers that remain consistent throughout the life of a process in the pod, regardless of whether the pod moves from one system to another. Since the pod namespace is distinct from the host's operating system namespace, the pod namespace can preserve this naming consistency for its processes even if the underlying operating system namespace changes, as may be the case in migrating processes from one machine to another. This consistency is essential to support process migration [16].

The pod private, virtual namespace enables secure isolation of applications by providing complete mediation to operating system resources. Pods can restrict what operating system resources are accessible within a pod by simply not providing identifiers to such resources within its namespace. A pod only needs to provide access to resources that are needed for running those processes within the pod. It does not need to provide access to all resources to support a complete operating system environment. An administrator can configure a pod in the same way one configures and installs applications on a regular machine. Pods enforce secure isolation to prevent exploited pods from being used to attack the underlying host or other pods on the system. Similarly, the secure isolation allows one to run multiple pods from different organizations, with different sets of users and administrators on a single host, while retaining the semantic of multiple distinct and individually managed machines.

For example, to provide a web server, one can easily setup a web server pod to only contain the files the web server needs to run and the content it wants to serve. The web server pod could have its own IP address, decoupling its network presence from the underlying system. The pod can have its network access limited to client-initiated connections using firewall software to restrict connections to the pod's IP address to only the ports served by the application running within this pod. If the web server application is compromised, the pod limits the ability of an attacker to further harm the system since the only resources he

has access to are the ones explicitly needed by the service. The attacker cannot use the pod to directly initiate connections to other systems to attack them since the pod is limited to client-initiated connections. Furthermore, there is no need to carefully disable other network services commonly enabled by the operating system to protect against the compromised pod since those services, and the core operating system itself, reside outside of the pod's context.

AutoPod Virtualization

To support the AutoPod abstraction design of secure and isolated namespaces on commodity operating systems, we employ a virtualization architecture that operates between applications and the operating system, without requiring any changes to applications or the operating system kernel. This virtualization layer is used to translate between the AutoPod namespaces and the underlying host operating system namespace. It protects the host operating system from dangerous privileged operations that might be performed by processes within the AutoPod, as well as protecting those processes from processes outside of the AutoPod.

Pods are supported using virtualization mechanisms that translate between the pod's resource identifiers and the operating system's resource identifiers. Every resource that a process in a pod accesses is through a *virtual private name* which corresponds to an operating system resource identified by a *physical name*. When an operating system resource is created for a process in a pod, such as with process or IPC key creation, instead of returning the corresponding physical name to the process, the pod virtualization layer catches the physical name value, and returns a virtual private name to the process. Similarly, any time a process passes a virtual private name to the operating system, the virtualization layer catches it and replaces it with the appropriate physical name.

The key pod virtualization mechanisms used are a system call interposition mechanism and the chroot utility with file system stacking to provide each pod with its own file system namespace that can be separate from the regular host file system. Pod virtualization support for migration is based on Zap [16]. We focus here on pod virtualization support for secure virtual machine isolation.

Because current commodity operating systems are not built to support multiple namespaces, AutoPod must take care of the security issues this causes. While chroot can give a set of processes a virtualized file system namespace, there are many ways to break out of the standard chrooted environment, especially if one allows the chroot system call to be used by processes in a pod. Pod file system virtualization enforces the chrooted environment and ensures that the pod's file system is only accessible to processes within the given pod by using a simple form of file system stacking to implement a barrier. File systems provide a permission function that determines if a process can access a file.

For example, if a process tries to access a file a few directories below the current directory, the permission function is called on each directory as well as the file itself in order. If any of the calls determine that the process does not have permission on a directory, the chain of calls end. Even if the permission function would determine that the process would have access to the file itself, it must have permission to traverse the directory hierarchy to the file to access it.

We implement a barrier by simply stacking a small pod-aware file system on top of the staging directory that overloads the underlying permission function to prevent processes running within the pod from accessing the parent directory of the staging directory, and to prevent processes running only on the host from accessing the staging directory. This effectively confines a process in a pod to the pod's file system by preventing it from ever walking past the pod's file system root.

While any network file system can be used with pods to support migration, we focus on NFS because it is the most commonly used network file system. Pods can take advantage of the user identifier (UID) security model in NFS to support multiple security domains on the same system running on the same operating system kernel. For example, since each pod can have its own private file system, each pod can have its own /etc/passwd file that determines its list of users and their corresponding UIDs. In NFS, the UID of a process determines what permissions it has in accessing a file.

By default, pod virtualization keeps process UIDs consistent across migration and keeps process UIDs the same in the pod and operating system namespaces. However, since the pod file system is separate from the host file system, a process running in the pod is effectively running in a separate security domain from another process with the same UID that is running directly on the host system. Although both processes have the same UID, each process is only allowed to access files in its own file system namespace. Similarly, multiple pods can have processes running on the same system with the same UID, but each pod effectively provides a separate security domain since the pod file systems are separate from one another.

The pod UID model supports an easy-to-use migration model when a user may be using a pod on a host in one administrative domain and then moves the pod to another. Even if the user has computer accounts in both administrative domains, it is unlikely that the user will have the same UID in both domains if they are administratively separate. Nevertheless, pods can enable the user to run the same pod with access to the same files in both domains.

Suppose the user has UID 100 on a machine in administrative domain A and starts a pod connecting to a file server residing in domain A. Suppose that all pod processes are then running with UID 100. When the user moves to a machine in administrative domain B where he has UID 200, he can migrate his pod to the new machine and continue running processes in the pod. Those processes can continue to run as UID 100 and continue to access the same set of files on the pod file server, even though the user's real UID has changed. This works, even if there's a regular user on the new machine with a UID of 100. While this example considers the case of having a pod with all processes running with the same UID, it is easy to see that the pod model supports pods that may have running processes with many different UIDs.

Because the root UID 0 is privileged and treated specially by the operating system kernel, pod virtualization treats UID 0 processes inside of a pod specially as well. AutoPod is required to do this to prevent processes running with privilege from breaking the pod abstraction, accessing resources outside of the pod, and causing harm to the host system. While a pod can be configured for administrative reasons to allow full privileged access to the underlying system, we focus on the case of pods for running application services which do not need to be used in this manner. Pods do not disallow UID 0 processes, which would limit the range of application services that could be run inside pods. Instead, pods provide restrictions on such processes to ensure that they function correctly inside of pods.

While a process is running in user space, its UID does not have any affect on process execution. Its UID only matters when it tries to access the underlying kernel via one of the kernel entry points, namely devices and system calls. Since a pod already provides a virtual file system that includes a virtual /dev with a limited set of secure devices, the device entry point is already secured. The only system calls of concern are those that could allow a root process to break the pod abstraction. Only a small number of system calls can be used for this purpose. These system calls are listed and described in further detail in the Appendix. Pod virtualization classifies these system calls into three classes.

The first class of system calls are those that only affect the host system and serve no purpose within a pod. Examples of these system calls include those that load and unload kernel modules or that reboot the host system. Since these system calls only affect the host, they would break the pod security abstraction by allowing processes within it to make system administrative changes to the host. System calls that are part of this class are therefore made inaccessible by default to processes running within a pod.

The second class of system calls are those that are forced to run unprivileged. Just like NFS, by default, squashes root on a client machine to act as user nobody, pod virtualization forces privileged processes to act as the nobody user when they want to make use of some system calls. Examples of these system calls include those that set resource limits and ioctl system calls. Since system calls such as setrlimit and nice can allow a privileged process to increase its resource limits beyond predefined limits imposed on pod processes, privileged processes are by default treated as unprivileged when executing these system calls within a pod. Similarly, the ioctl system call is a system call multiplexer that allows any driver on the host to effectively install its own set of system calls. Since the ability to audit the large set of possible system calls is impossible given that pods may be deployed on a wide range of machine configurations that are not controlled by the AutoPod system, pod virtualization conservatively treats access to this system call as unprivileged by default.

The final class of system calls are calls that are required for regular applications to run, but have options that will give the processes access to underlying host resources, breaking the pod abstraction. Since these system calls are required by applications, the pod checks all their options to ensure that they are limited to resources that the pod has access to, making sure they are not used in a manner that breaks the pod abstraction. For example, the mknod system call can be used by privileged processes to make named pipes or files in certain application services. It is therefore desirable to make it available for use within a pod. However, it can also be used to create device nodes that provide access to the underlying host resources. To limit how the system call is used, the pod system call interposition mechanism checks the options of the system call and only allows it to continue if it is not trying to create a device.

Migration Across Different Kernels

To maintain application service availability without losing important computational state as a result of system downtime due to operating system upgrades, AutoPod provide a checkpoint-restart mechanism that allows pods to be migrated across machines running different operating system kernels. Upon completion of the upgrade process, the respective AutoPod and its applications are restored on the original machine. We assume here that any kernel security holes on the unpatched system have not yet been exploited on the system; migrating across kernels that have already been compromised is beyond the scope of this paper. We also limit our focus to migrating between machines with a common CPU architecture with kernel differences that are limited to maintenance and security patches. These patches often correspond to changes in the minor version number of the kernel. For example, the Linux 2.4 kernel has nearly thirty minor versions. Even within minor version changes, there can be significant changes in kernel code. Table 1 shows the

number of files that have been changed in various subsystems of the Linux 2.4 kernel across different minor versions. For example, all of the files for the VM subsystem were changed since extensive modifications were made to implement a completely new page replacement mechanism in Linux.

Many of the Linux kernel patches contain security vulnerability fixes, which are typically not separated out from other maintenance patches. We similarly limit our focus to where the application's execution semantics, such as how threads are implemented and how dynamic linking is done, do not change. On the Linux kernels this is not an issue as all these semantics are enforced by user-space libraries. Whether one uses kernel or user threads, or how libraries are dynamically linked into a process is all determined by the respective libraries on the file system. Since the pod has access to the same file system on whatever machine it is running on, these semantics stay the same.

To support migration across different kernels, AutoPod use a checkpoint-restart mechanism that employs an intermediate format to represent the state that needs to be saved on checkpoint. On checkpoint, the intermediate format representation is saved and digitally signed to enable the restart process to verify the integrity of the image. Although the internal state that the kernel maintains on behalf of processes can be different across different kernels, the high-level properties of the process are much less likely to change. We capture the state of a process in terms of higher-level semantic information specified in the intermediate format rather than kernel specific data in native format to keep the format portable across different kernels.

For example, the state associated with a UNIX socket connection consists of the directory entry of the UNIX socket file, its superblock information, a hash key, and so on. It may be possible to save all of this state in this form and successfully restore on a different machine running the same kernel. But this representation of a UNIX socket connection state is of limited portability across different kernels. A different high-level representation consisting of a four tuple, virtual source PID, source FD, virtual destination PID, destination FD is highly portable. This is because the semantics of a process identifier and a file descriptor are typically standard across different kernels, especially across minor version differences.

The intermediate representation format used by AutoPod for migration is chosen such that it offers the degree of portability needed for migrating between different kernel minor versions. If the representation of state is too high-level, the checkpoint-restart mechanism could become complicated and impose additional overhead. For example, the AutoPod system saves the address space of a process in terms of discrete memory regions called virtual memory (VM) areas. As an alternative, it may be possible to save the contents of a process's address space and denote the characteristics of various portions of it in more abstract terms. However, this would call for an unnecessarily complicated interpretation scheme and make the implementation inefficient. The VM area abstraction is standard across major Linux kernel revisions. AutoPod view the VM area abstraction as offering sufficient portability in part because the organization of a process's address space in this manner has been standard across all Linux kernels and has never changed.

AutoPod further support migration across different kernels by leveraging higher-level native kernel services to transform intermediate representation of the checkpointed image into an internal representation suitable for the target kernel. Continuing with the previous example, AutoPod restore a UNIX socket connection using high-level kernel functions as follows. First, two new processes are created with virtual PIDs as specified in the four tuple. Then, each one creates a UNIX socket with the specified file descriptor and one socket is made to connect to the other. This procedure effectively recreates the original UNIX socket connection without depending on many kernel internal details.

This use of high-level functions helps in general portability of using AutoPod for migration. Security patches and minor version kernel revisions commonly involve modifying the internal details of the kernel while high-level primitives remain unchanged. As such services are usually made available to kernel modules, the AutoPod system is able to perform cross-kernel migration without requiring modifications to the kernel code.

The AutoPod checkpoint-restart mechanism is also structured in such a way to perform its operations

Type	2.4.1	2.4.29	Modified	Unmodified	% Unmodified
Drivers	2623	3784	1742	501	13.2
Arch	123	128	93	22	17.1
FS	536	894	410	59	6.6
Network	461	600	338	84	9.4
Core Kernel	27	27	24	3	11.1
VM	21	20	20	0	0
IPC	6	6	5	1	16.6

Table 1: Kernel file changes within the Linux 2.4 series for i386.

when processes are in a state that checkpointing can avoid depending on many low-level kernel details. For example, semaphores typically have two kinds of state associated with each of them: the value of the semaphore and the wait queue of processes waiting to acquire the corresponding semaphore lock. In general, both of these pieces of information have to be saved and restored to accurately reconstruct the semaphore state. Semaphore values can be easily obtained and restored through GETALL and SETALL parameters of the semctl system call. But saving and restoring the wait queues involves manipulating kernel internals directly. The AutoPod mechanism avoids having to save the wait queue information by requiring that all the processes be stopped before taking the checkpoint. When a process waiting on a semaphore receives a stop signal, the kernel immediately releases the process from the wait queue and returns EINTR. This ensures that the semaphore wait queues are always empty at the time of checkpoint so that they do not have to be saved.

While AutoPod can abstract and manipulate most process state in higher-level terms using higher-level kernel services, there are some parts that not amenable to a portable intermediate representation. For instance, specific TCP connection states like timestamp values and sequence numbers, which do not have a high-level semantic value, have to be saved and restored to maintain a TCP connection. As this internal representation can change, its state needs to be tracked across kernel versions and security patches. Fortunately, there is usually an easy way to interpret such changes across different kernels because networking standards such as TCP do not change often. Across all of the Linux 2.4 kernels, there was only one change in TCP state that required even a small modification in the AutoPod migration mechanism. Specifically, in the Linux 2.4.14 kernel, an extra field was added to TCP connection state to address a flaw in the existing syncookie mechanism. If configured into the kernel, syncookies protect an Internet server against a synflood attack. When migrating from an earlier kernel to a Linux-2.4.14 or later version kernel, the AutoPod system initializes the extra field in such a way that the integrity of the connection is maintained. In fact, this was the only instance across all of the Linux 2.4 kernel versions where an intermediate representation was not possible and the internal state had changed and had to be accounted for.

To provide proper support for AutoPod virtualization when migrating across different kernels, we must ensure that that any changes in the system call interfaces are properly accounted for. As AutoPod has a virtualization layer using system call interposition mechanism for maintaining namespace consistency, a change in the semantics for any system call intercepted by AutoPod could be an issue in migrating across different kernel versions. But such changes usually do not occur as it would require that the libraries be rewritten. In other words, AutoPod virtualization is protected from such changes in a similar way as legacy applications are protected. However, new system calls could be added from time to time. For instance, across all Linux 2.4 kernels to date, there were two new system calls, gettid and tkill for querying the thread identifier and for sending a signal to a particularly thread in a thread group, respectively, which needed to be accounted for to properly virtualize AutoPod across kernel versions. As these system calls take identifier arguments, they were simply intercepted and virtualized.

Autonomic System Status Service

AutoPod provides a generic autonomic framework for managing system state. The framework is able to monitor multiple sources for information and can use this information to make autonomic decisions about when to checkpoint pods, migrate them to other machines, and restart them. While there are many items that can be monitored, our service monitors two items in particular. First, it monitors the vendor's software security update repository to ensure that the system stays up to date with the latest security patches. Second, it monitors the underlying hardware of the system to ensure that an imminent fault is detected before the fault occurs and corrupts application state. By monitoring these two sets of information, the autonomic system status service is able to reboot or shutdown the computer, while checkpointing or migrating the processes. This helps ensure that data is not lost or corrupted due to a forced reboot or a hardware fault propagating into the running processes.

Many operating system vendors provide their users with the ability to automatically check for system updates and to download and install them when they become available. Example of these include Microsoft's Windows Update service, as well as Debian based distribution's security repositories. Users are guaranteed that the updates one gets through these services are genuine because they are verified through cryptographic signed hashes that verify the contents as coming from the vendors. The problem with these updates is that some of them require machine reboots; In the case of Debian GNU/Linux this is limited to kernel upgrades. We provide a simple service that monitors these security update repositories. The autonomic service simply downloads all security updates, and by using the pod's checkpoint/restart mechanism enables the security updates that need reboots to take effect without disrupting running applications and causing them to lose state.

Commodity systems also provide information about the current state of the system that can indicate if the system has an imminent failure on its hands. Subsystems, such as a hard disk's Self-Monitoring Analysis Reporting Technology (SMART), let an autonomic service monitor the system's hardware state. SMART

provides diagnostic information, such as temperature and read/write error rates, on the hard drives in the system that can indicate if the hard disk is nearing failure. Many commodity computer motherboards also have the ability to measure CPU and case temperature, as well as the speeds of the fans that regulate those temperatures. If temperature in the machine rises too high, hardware in the machine can fail catastrophically. Similarly, if the fans fail and stop spinning, the temperature will likely rise out of control. Our autonomic service monitors these sensors and if it detects an imminent failure, will attempt to migrate an AutoPod to a cooler system, as well as shutdown the machine to prevent the hardware from being destroyed.

Many administrators use an uninterruptible power supply to avoid having a computer lose or corrupt data in the event of a power loss. While one can shutdown a computer when the battery backup runs low, most applications are not written to save their data in the presence of a forced shutdown. AutoPod, on the other hand, monitors UPS status and if the battery backup becomes low can quickly checkpoint the pod's state to avoid any data loss when the computer is forced to shutdown.

Similarly, the operating system kernel on the machine monitors the state of the system, and if irregular conditions occur, such as DMA timeout or needing to reset the IDE bus, will log this occurrence. Our autonomic service monitors the kernel logs to discover these irregular conditions. When the hardware monitoring systems or the kernel logs provide information about possible pending system failures, the autonomic service checkpoints the pods running on the system, and migrates them to a new system to be restarted on. This ensures state is not lost, while informing system administrators the a machine needs maintenance.

Many policies can be implemented to determine which system a pod should be migrated to while a machine needs maintenance. Our autonomic service uses a simple policy of allowing a pod to be migrated around a specified set of clustered machines. The autonomic service gets reports at regular intervals from the other machines' autonomic services that reports each machine's load. If the autonomic service decides that it must migrate a pod, it chooses the machine in its cluster that has the lightest load.

Security Analysis

Saltzer and Schroeder [24] describe several principles for designing and building secure systems. These include:

- **Economy of mechanism**: Simpler and smaller systems are easier to understand and ensure that they do not allow unwanted access.
- **Complete mediation**: Systems should check every access to protected objects.
- **Least privilege**: A process should only have access to the privileges and resources it needs to do its job.
- **Psychological acceptability**: If users are not willing to accept the requirements that the security system imposes, such as very complex passwords that the users are forced to write down, security is impaired. Similarly, if using the system is too complicated, users will misconfigure it and end up leaving it wide open.
- **Work factor**: Security designs should force an attacker to have to do extra work to break the system. The classic quantifiable example is when one adds a single bit to an encryption key, one doubles the key space an attacker has to search.

AutoPod is designed to satisfy these five principles. AutoPod provides economy of mechanism using a thin virtualization layer based on system call interposition and file system stacking that only adds a modest amount of code to a running system. Furthermore, AutoPod changes neither applications nor the underlying operating system kernel. The modest amount of code to implement AutoPod makes the system easier to understand. Since the AutoPod security model only provides resources that are physically within the environment, it is relatively easy to understand the security properties of resource access provided by the model.

AutoPod provides for complete mediation of all resources available on the host machine by ensuring that all resources accesses occur through the pod's virtual namespace. Unless a file, process, or other operating system resource was explicitly placed in the pod by the administrator or created within the pod, AutoPod's virtualization will not allow a process within a pod to access the resource.

AutoPod provides a least privilege environment by enabling an administrator to only include the data necessary for each service. AutoPod can provide separate pods for individual services so that separate services are isolated and restricted to the appropriate set of resources. Even if a service is exploited, AutoPod will limit the attacker to the resources the administrator provided for that service. While one can achieve similar isolation by running each individual service on a separate machine, this leads to inefficient use of resources. AutoPod maintains the same least privilege semantic of running individual services on separate machines, while making efficient use of machine resources at hand. For instance, an administrator could run MySQL and Exim mail transfer services on a single machine, but within different pods. If the Exim pod gets exploited, the pod model ensures that the MySQL pod and its data will remain isolated from the attacker.

AutoPod provides psychological acceptability by leveraging the knowledge and skills system administrators already use to setup system environments. Because pods provide a virtual machine model, administrators can use their existing knowledge and skills to run their services within pods. This differs

from other least privilege architectures that force an administrator to learn new principles or complicated configuration languages that require a detailed understanding of operating system principles.

AutoPod increases the work factor required to compromise a system by not making available the resources that attackers depend on to harm a system once they have broken in. For example, services like mail delivery do not depend on having access to a shell. By not including a shell program within a mail delivery AutoPod, one makes it difficult for an attacker to get a root shell that they would use to further their attacks. Similarly, the fact that one can migrate a system away from a host that is vulnerable to attack increases the work an attacker would have to do to make services unavailable.

AutoPod Examples

We briefly describe two examples that help illustrate how AutoPod can be used to improve application availability for different application scenarios. The application scenarios are system services, such as e-mail delivery and desktop computing. In both cases we describe the architecture of the system and show how it can be run within AutoPod, enabling administrators to reduce downtime in the face of machine maintenance. We also discuss how a system administrator can setup and use pods.

System Services

Administrators like to run many services on a single machine. By doing this, they are able to benefit from improved machine utilization, but at the same time give each service access to many resources they do not need to perform their job. A classic example of this is e-mail delivery. E-mail delivery services, such as Exim, are often run on the same system as other Internet services to improve resource utilization and simplify system administration through server consolidation. However, services such as Exim have been easily exploited by the fact that they have access to system resources, such as a shell program, that they do not need to perform their job.

For e-mail delivery, AutoPod can isolate e-mail delivery to provide a significantly higher level of security in light of the many attacks on mail transfer agent vulnerabilities that have occurred. Consider isolating an Exim service, the default Debian mail transfer agent, installation. Using AutoPod, Exim can execute in a resource restricted pod, which isolates e-mail delivery from other services on the system. Since pods allow one to migrate a service between machines, the e-mail delivery pod is migratable. If a fault is discovered in the underlying host machine, the e-mail delivery service can be moved to another system while the original host is patched, preserving the availability of the e-mail service.

With this e-mail delivery example, a simple system configuration can prevent the common buffer overflow exploit of getting the privileged server to execute a local shell. This is done by just removing shells from within the Exim pod, thereby limiting the amateur attacker's ability to exploit flaws while requiring very little additional knowledge about how to configure the service. AutoPod can further automatically monitor system status and checkpoint the Exim pod if a fault is detected to ensure that no data is lost or corrupted. Similarly, in the event that a machine has to be rebooted, the service can automatically be migrated to a new machine to avoid any service downtime.

A common maintenance problem system administrators face is that forced machine downtime, for example due to reboots, can cause a service to be unavailable for a period of time. A common way to avoid this problem is to throw multiple machines at the problem. By providing the service through a cluster of machines, system administrators can upgrade the individual machines in a rolling manner. This enables system administrators to upgrade the systems providing the service while keeping the service available. The problem with this solution is that system administrators need to throw more machines at the problem than they might need to provide the service effectively, thereby increasing management complexity as well as cost.

AutoPod in conjunction with hardware virtual machine monitors improves this situation immensely. Using a virtual machine monitor to provide two virtual machines on a single host, AutoPod can then run a pod within a virtual machine to enable a single node maintenance scenario that can decrease costs as well management complexity. During regular operation, all application services run within the pod on one virtual machine. When one has to upgrade the operating system in the running virtual machine, one brings the second virtual machine online and migrates the pod to the new virtual machine.

Once the initial virtual machine is upgraded and rebooted, the pod can be migrated back to it. This reduces costs as only a single physical machine is needed. This also reduces management complexity as only one virtual machine is in use for the majority of the time the service is in operation. Since AutoPod runs unmodified applications, any application service that can be installed can make use of AutoPod's ability to provide general single node maintenance.

Desktop Computing

As personal computers have become more ubiquitous in large corporate, government, and academic organizations, the total cost of owning and maintaining them is becoming unmanageable. These computers are increasingly networked which only complicates the management problem. They need to be constantly patched and upgraded to protect them, and their data, from the myriad of viruses and other attacks commonplace in today's networks.

To solve this problem, many organizations have turned to thin-client solutions such as Microsoft's Windows Terminal Services and Sun's Sun Ray. Thin clients give administrators the ability to centralize many of their administrative duties as only a single computer or a cluster of computers needs to be maintained in a central location, while stateless client devices are used to access users' desktop computing environments. While thin-client solutions provide some benefits for lowering administrative costs, this comes at the loss of semantics users normally expect from a private desktop. For instance, users who use their own private desktop expect to be isolated from their coworkers. However, in a shared thin-client environment, users share the same machine. There may be many shared files and a user's computing behavior can impact the performance of other users on the system.

While a thin-client environment minimizes the machines one has to administrate, the centralized servers still need to be administrated, and since they are more highly utilized, management becomes more difficult. For instance, on a private system one only has to schedule system maintenance with a single user, as reboots will force the termination of all programs running on the system. However, in a thin-client environment, one has to schedule maintenance with all the users on the system to avoid having them lose any important data.

AutoPod enables system administrators to solve these problems by allowing each user to run a desktop session within a pod. Instead of users directly sharing a single file system, AutoPod provides each pod with a composite of three file systems: a shared read-only file system of all the regular system files users expect in their desktop environments, a private writable file system for a user's persistent data, and a private writable file system for a user's temporary data. By sharing common system files, AutoPod provides centralization benefits that simplify system administration. By providing private writable file systems for each pod, AutoPod provides each user with privacy benefits similar to a private machine.

Coupling AutoPod virtualization and isolation mechanisms with a migration mechanism can provide scalable computing resources for the desktop and improve desktop availability. If a user needs access to more computing resources, for instance while doing complex mathematical computations, AutoPod can migrate that user's session to a more powerful machine. If maintenance needs to be done on a host machine, AutoPod can migrate the desktop sessions to other machines without scheduling downtime and without forcefully terminating any programs users are running.

Setting Up and Using AutoPod

To demonstrate how simple it is to setup a pod to run within the AutoPod environment, we provide a step by step walkthrough on how one would create a new pod that can run the Exim mail transfer agent. Setting up AutoPod to provide the Exim pod on Linux is straightforward and leverages the same skill set and experience system administrators already have on standard Linux systems. AutoPod is started by loading its kernel module into a Linux system and using its user-level utilities to setup and insert processes into a pod.

Creating a pod's file system is the same as creating a chroot environment. Administrators that have experience creating a minimal environment, that just contains the application they want to isolate, do not need to do any extra work. However, many administrators do not have experience creating such an environment and therefore need an easy way to create an environment to run their application in. These administrators can take advantage of Debian's debootstrap utility that enables a user to quickly setup an environment that's the equivalent of a base Debian installation. An administrator would do a `debootstrap stable /pod` to install the most recently released Debian system into the `/pod` directory. While this will also include many packages that are not required by the installation, it provides a small base to work from. An administrator can remove packages, such as the installed mail transfer agent, that are not needed.

To configure Exim, an administrator edits the appropriate configuration files within the `/pod/etc/exim4/` directory. To run Exim in a pod, an administrator does `mount -o bind /pod /autopod/exim/root` to loopback mount the pod directory onto the staging area directory where AutoPod expects it. `autopod add exim` is used to create a new pod named exim which uses `/autopod/exim/root` as the root for its file system. Finally, `autopod addproc exim /usr/sbin/exim4` is used to start Exim within the pod by executing the `/usr/sbin/exim4` program, which is actually located at `/autopod/exim/root/usr/sbin/exim4`.

AutoPod isolates the processes running within a pod from the rest of the system, which helps contain intrusions if they occur. However, since a pod does not have to be maintained by itself, but can be maintained in the context of a larger system, one can also prune down the environment and remove many programs that an attacker could use against the system. For instance, if an Exim pod has no need to run any shell scripts, there is no reason an administrator has to leave programs such as `/bin/bash`, `/bin/sh` and `/bin/dash` within the environment. One issue is that these programs are necessary if the administrator wants to be able to simply upgrade the package in the future via normal Debian methods. Since it is simple to recreate the environment, one approach would be to remove all the programs that are not wanted within the environment and recreate the environment when an upgrade is needed. Another approach would be to move those programs outside of the pod, such as by creating a `/pod-backup` directory. To upgrade the pod using the normal Debian package upgrade methods, the programs can then be moved back into the pod file system.

If an administrator wants to manually reboot the system without killing the processes within this Exim pod, the administrator can first checkpoint the pod to disk by running autopod checkpoint exim -o /exim.pod, which tells AutoPod to checkpoint the processes associated with the exim pod to the file /exim.pod. The system can then be rebooted, potentially with an updated kernel. Once it comes back up, the pod can be restarted from the /exim.pod file by running autopod restart exim -i /exim.pod. These mechanisms are the same as those used by the AutoPod system status service for controlling the checkpointing and migration of pods.

Standard Debian facilities for installing packages can be used for running other services within a pod. Once the base environment is setup, an administrator can chroot into this environment by running chroot /pod to continue setting it up. By editing the /etc/apt/sources.list file appropriately and running apt-get update, an administrator will be able to install any Debian package into the pod. In the Exim example, Exim does not need to be installed since it is the default MTA and already included in the base Debian installation. If one wanted to install another MTA, such as Sendmail, one could run apt-get install sendmail, which will download Sendmail and all the packages needed to run it. This will work for any service available within Debian. An administrator can also use the dpkg --purge option to remove packages that are not required by a given pod. For instance, in running an Apache web server in a pod, one could remove the default Exim mail transfer agent since it is not needed by Apache.

Experimental Results

We implemented AutoPod as a loadable kernel module in Linux, that requires no changes to the Linux kernel, as well as a user space system status monitoring service. We present some experimental results using our Linux prototype to quantify the overhead of using AutoPod on various applications. Experiments were conducted on a trio of IBM Netfinity 4500R machines, each with a 933 Mhz Intel Pentium-III CPU, 512 MB RAM, 9.1 GB SCSI HD and a 100 Mbps Ethernet connected to a 3Com Superstack II 3900 switch. One of the machines was used as an NFS server from which directories were mounted to construct the virtual file system for the AutoPod on the other client systems. The clients ran different Linux distributions and kernels, one machine running Debian Stable with a Linux 2.4.5 kernel and the other running Debian Unstable with a Linux 2.4.18 kernel.

To measure the cost of AutoPod virtualization, we used a range of micro benchmarks and real application workloads and measured their performance on our Linux AutoPod prototype and a vanilla Linux system. Table 2 shows the seven micro-benchmarks and four application benchmarks we used to quantify AutoPod virtualization overhead as well as the results for a vanilla Linux system. To obtain accurate measurements, we rebooted the system between measurements. Additionally, the system call micro-benchmarks directly used the TSC register available on Pentium CPUs to record timestamps at the significant measurement events. Each timestamp's average cost was 58 ns. The files for the benchmarks were stored on the NFS Server. All of these benchmarks were performed in a chrooted environment on the NFS client machine running Debian Unstable with a Linux 2.4.18 kernel. Figure 4 shows the results of running the benchmarks under both configurations, with the vanilla Linux configuration normalized to one. Since all benchmarks measure the time to run the benchmark, a small number is better for all benchmarks results.

The results in Figure 2 show that AutoPod virtualization overhead is small. AutoPod incurs less than 10% overhead for most of the micro-benchmarks and less than 4% overhead for the application workloads. The overhead for the simple system call getpid benchmark is only 7% compared to vanilla Linux, reflecting the fact that AutoPod virtualization for these kinds of system calls only requires an extra procedure call and

Name	Description	Linux
getpid	average getpid runtime	350 ns
ioctl	average runtime for the FIONREAD ioctl	427 ns
shmget-shmctl	IPC Shared memory segment holding an integer is created and removed	3361 ns
semget-semctl	IPC Semaphore variable is created and removed	1370 ns
fork-exit	process forks and waits for child which calls exit immediately	44.7 us
fork-sh	process forks and waits for child to run /bin/sh to run a program that prints "hello world" then exits	3.89 ms
Apache	Runs Apache under load and measures average request time	1.2 ms
Make	Linux Kernel compile with up to 10 process active at one time	224.5 s
Postmark	Use Postmark Benchmark to simulate Exim performance	.002 s
MySQL	"TPC-W like" interactions benchmark	8.33 s

Table 2: Application benchmarks.

a hash table lookup. The most expensive benchmarks for AutoPod is semget+semctl which took 51% longer than vanilla Linux. The cost reflects the fact that our untuned AutoPod prototype needs to allocate memory and do a number of namespace translations.

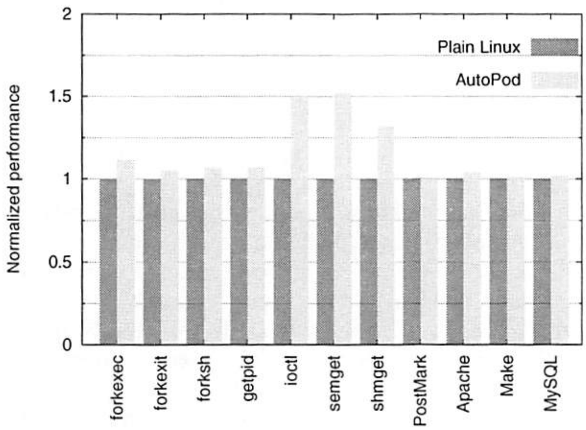


Figure 2: AutoPod virtualization overhead.

The ioctl benchmark also has high overhead, because of the 12 separate assignments it does to protect the call against malicious root processes. This is large compared to the simple FIONREAD ioctl that just performs a simple dereference. However, since the ioctl is simple, we see that it only adds 200 ns of overhead over any ioctl. For real applications, the most overhead was only four percent which was for the Apache workload, where we used the http_load benchmark [18] to place a parallel fetch load on the server with 30 clients fetching at the same time. Similarly, we tested MySQL as part of a web-commerce scenario outlined by TPC-W with a bookstore servlet running on top of Tomcat with a MySQL back-end. The AutoPod overhead for this scenario was less than 2% versus vanilla Linux.

To measure the cost of AutoPod migration and demonstrate the ability of AutoPod to migrate real applications, we migrated the three application scenarios; an email delivery service using Exim and Procmail, a web content delivery service using Apache and MySQL, and a KDE desktop computing environment. Table 3 described the configurations of the application scenarios we migrated, as well as showing the time it takes to startup on a regular Linux system. To demonstrate our AutoPod prototype's ability to migrate across Linux kernels with different minor versions, we checkpointed each application workload on the 2.4.5 kernel client machine and restarted it on the 2.4.18 kernel machine. For these experiments, the workloads were checkpointed to and restarted from local disk.

Case	Check Point	Restart	Size	Compr'd
E-mail	11 ms	14 ms	284 KB	84 KB
Web	308 ms	47 ms	5.3 MB	332 KB
Desktop	851 ms	942 ms	35 MB	8.8 MB

Table 4: AutoPod migration costs.

Table 4 shows the time it took to checkpoint and restart each application workload. In addition to these, migration time also has to take into account network transfer time. As this is dependent on the transport medium, we include the uncompressed and compressed checkpoint image sizes. In all cases, checkpoint and restart times were significantly faster than the regular startup times listed in Table 5, taking less than a second for both operations, even when performed on separate machines or across a reboot. We also show that the actual checkpoint images that were saved were modest in size for complex workloads. For example, the Desktop pod had over 30 different processes running, providing the KDE desktop applications, as well as substantial underlying window system infrastructure, including inter-application sharing, a rich desktop interface managed by a window manager with a number of applications running in a panel such as the clock. Even with all these applications running, they checkpoint to a very reasonable 35 MB uncompressed for a full desktop

Name	Applications	Normal Startup
E-mail	Exim 3.36	504 ms
Web	Apache 1.3.26 and MySQL 4.0.14 .	2.1 s
Desktop	Xvnc – VNC 3.3.3r2 X Server	19 s
	KDE – Entire KDE 2.2.2 environment, including window manager, panel and assorted background daemon and utilities	
	SSH – openssh 3.4p1 client inside a KDE konsole terminal connected to a remote host	
	Shell – The Bash 2.05a shell running in a konsole terminal	
	KGhostView – A PDF viewer with a 450 KB 16 page PDF file loaded.	
	Konqueror – A modern standards compliant web browser that is part of KDE	
	KOffice – The KDE word processor and spreadsheet programs	

Table 3: Application scenarios.

environment. Additionally, if one needed to transfer the checkpoint images over a slow link, Table 6 shows that they can be compressed very well with the bzip2 compression program.

Related Work

Virtual machine monitors (VMMs) have been used to provide secure isolation [4, 29, 30], and have also been used to to migrate an entire operating system environment [25]. Unlike AutoPod, standard VMMs decouple processes from the underlying machine hardware, but tie them to an instance of an operating system. As a result, VMMs cannot migrate processes apart from that operating system instance and cannot continue running those processes if the operating system instance ever goes down, such as during security upgrades. In contrast, AutoPod decouples process execution from the underlying operating system which allows it to migrate processes to another system when an operating system instance is upgraded. VMMs have been proposed to support online maintenance of systems [14] by having a microvisor that supports at most two virtual machines running on the machine at the same time, effectively giving each physical machine the ability to act as its own hot spare. However, this proposal explicitly depends on AutoPod migration functionality yet does not provide it.

A number of other approaches have explored the idea of virtualizing the operating system environment to provide application isolation. FreeBSD's Jail mode [10] provides a chroot like environment that processes can not break out of. However, since Jail is limited in what it can do, such as the fact it does not allow IPC within a jail [9] many real world application will not work. More recently, Linux Vserver [1] and Solaris Zones [19] offer a similar virtual machine abstraction as AutoPod pods, but require substantial in-kernel modifications to support the abstraction. They do not provide isolation of migrating applications across independent machines, and have no support for maintaining application availability in the presence of operating system maintenance and security upgrades.

Many systems have been proposed to support process migration [2, 3, 6, 7, 8, 13, 15, 17, 20, 21, 23, 26], but do not allow migration across independent machines running different operating system versions. TUI [27] provides support for process migration across machines running different operating systems and hardware architectures. Unlike AutoPod, TUI has to compile applications on each platform using a special compiler and does not work with unmodified legacy applications. AutoPod builds on a pod abstraction introduced in Zap [16] to support transparent migration across systems running the same kernel version. Zap does not address security issues or heterogeneous migration. AutoPod goes beyond Zap in providing a complete, secure virtual machine abstraction for isolating processes, finer-grain mechanisms for isolating application components, and transparent migration across minor kernel versions, which is essential for providing application availability in the presence of operating system security upgrades.

Replication in clustered systems can provide the ability to do rolling upgrades. By leveraging many nodes, individual nodes can be taken down for maintenance, without significantly impacting the load the cluster can handle. For example, web content is commonly delivered by multiple web servers behind a front end manager. This front end manager enables an administrator to bring down back end web servers for maintenance as it will only direct requests to the active web servers. This simple solution is effective because it is easy to replicate web servers to serve the same content. While this model works fine for web server loads, as the individual jobs are very short, it does not work for long running jobs, such as a user's desktop. In the web server case, replication and upgrades are easy to do since only one web server is used to serve any individual request and any web server can be used to serve any request. For long running stateful applications, such as a user's desktop, requests cannot be arbitrarily redirected to any desktop computing environment as each user's desktop session is unique. While specialized hardware support could be used to keep replicas synchronized, by having all of them process all operations, this is prohibitively expensive for most workloads and does not address the problem of how to resynchronize the replicas in the presence of rolling upgrades.

Another possible solution to this problem is allowing the kernel to be hot plugable. While micro-kernels are not prevalent, they contain this ability to upgrade their parts on the fly. More commonly, many modern monolithic kernels have kernel modules that can be inserted and removed dynamically. This can allow one to upgrade parts of a monolithic kernel without requiring any reboots. The Nooks [28] system extends this concept by enabling kernel drivers and other kernel functionality, such as file systems, to be isolated into their own protection domain to help isolate faults in kernel code and provide a more reliable system. However, in all of these cases, their is still a base kernel on the machine that cannot be replaced without a reboot. If one has to replace that part, all data would be lost.

The K42 operating system has the ability to be dynamically updated [5]. This functionality enables software patches to be applied to a running kernel even in the presence of data structure changes. However, it requires a completely new operating system design and does not work with any commodity operating system. Even on K42, it is not yet possible to upgrade the kernel while running realistic application workloads.

Conclusions

The AutoPod system provides an operating system virtualization layer that decouples process execution from the underlying operating system, by running the process within a pod. Pods provide an easy-to-use

lightweight virtual machine abstraction that can securely isolate individual applications without the need to run an operating system instance in the pod. Furthermore, AutoPod can be transparently migrate isolated applications across machines running different operating system kernel versions. This enables security patches to be applied to operating systems in a timely manner with minimal impact on the availability of application services. It also preserves secure isolation of untrusted applications in the presence of migration.

We have implemented AutoPod on Linux without requiring any application or operating system kernel changes. We demonstrated how pods can be used to enable autonomic machine maintenance and increase availability for a range of applications, including e-mail delivery, web servers with databases and desktop computing. Our measurements on real world applications demonstrate that AutoPod imposes little overhead, provides sub-second suspend and resume times that can be an order of magnitude faster than starting applications after a system reboot, and enables systems to autonomously stay updated with relevant maintenance and security patches, while ensuring no loss of data and minimizing service disruption.

Acknowledgments

Matt Selsky contributed to the architecture of the isolation mechanism. Dinesh Subhraveti contributed to the implementation of the process migration mechanism. This work was supported in part by NSF grants CNS-0426623 and ANI-0240525, and an IBM SUR Award.

Author Information

Shaya Potter is a Ph.D. candidate in Columbia University's Computer Science department. His research interests are focused around improving computer usage for users and administrators through virtualization and process migration technologies. He received his B.A. from Yeshiva University and his M.S. and M.Phil degrees from Columbia University, all in Computer Science. Reach him electronically at spotter@cs.columbia.edu .

Jason Nieh is an Associate Professor of Computer Science at Columbia University and Director of the Network Computing Laboratory. He is also the technical adviser for nine States on the Microsoft Antitrust Settlement. He received his B.S. from MIT and his M.S. and Ph.D. from Stanford University, all in Electrical Engineering. Reach him electronically at nieh@cs.columbia.edu .

Bibliography

[1] *Linux VServer Project,* http://www.linux-vserver.org/ .

[2] Artsy, Y, Y. Chang, and R. Finkel, "Interprocess communication in charlotte," *IEEE Software*, pages 22-28, January, 1987.

[3] Barak, A. and R. Wheeler, "MOSIX: An Integrated Multiprocessor UNIX," *Proceedings of the USENIX Winter 1989 Technical Conference*, pp. 101-112, San Diego, CA, February, 1989.

[4] Barham, P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauery, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, October, 2003.

[5] Baumann, A., J. Appavoo, D. D. Silva, J. Kerr, O. Krieger, and R. W. Wisniewski, "Providing dynamic update in an operating system," *USENIX 2004*, pp. 279-291, Anaheim, California, April, 2005.

[6] Casas, J., D. Clark, R. Konuru, S. Otto, R. Prouty, and J. Walpole, "MPVM: A migration transparent version of PVM," *Computing Systems*, Vol. 8, Num. 2, pp. 171-216, 1995.

[7] Cheriton, D., "The V distributed system," *Communications of the ACM*, Vol. 31, Num. 3, pp. 314-333, March, 1988.

[8] Douglis, F. and J. Ousterhout, "Transparent process migration: Design alternatives and the sprite implementation," *Software – Practice and Experience*, Vol. 21, Num. 8, pp. 757-785, August, 1991.

[9] FreeBSD Project, *Developer's handbook,* http://www.freebsd.org/doc/en_US.ISO8859-1/books/developers-handbook/secure-chroot.html .

[10] Kamp, P.-H. and R. N. M. Watson, "Jails: Confining the omnipotent root," *2nd International SANE Conference*, MECC, Maastricht, The Netherlands, May, 2000.

[11] Kephart, J. O., and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, pages 41-50, January, 2003.

[12] LaMacchia, B., Personal Communication, January, 2004.

[13] Litzkow, M., T. Tannenbaum, J. Basney, and M. Livny, *Checkpoint and migration of unix processes in the condor distributed processing system*, Technical Report 1346, University of Wisconsin-Madison Computer Sciences, April, 1997.

[14] Lowell, D. E., Y. Saito, and E. J. Samberg, "Devirtualizable virtual machines enabling general, single-node, online maintenance," *Eleventh International Conference on Architectural Support for Programming Languages and Operating Systems*, October, 2004.

[15] Mullender, S. J., G. v. Rossum, A. S. Tanenbaum, R. v. Renesse, and H. v. Staveren, "Amoeba: a distributed operating system for the 1990s," *IEEE Computer*, Vol. 23, Num. 5, pp. 44-53, May, 1990.

[16] Osman, S. D. Subhraveti, G. Su, and J. Nieh, "The Design and Implementation of Zap: A System for

Migrating Computing Environments," *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, December, 2002.

[17] Plank, J. S., M. Beck, G. Kingsley, and K. Li, "Libckpt: Transparent checkpointing under UNIX," *Proceedings of Usenix Winter 1995 Technical Conference*, pp. 213-223, New Orleans, LA, January, 1995.

[18] Poskanzer, J., http://www.acme.com/software/http_load/ .

[19] Price, D. and A. Tucker, "Solaris zones: Operating system support for consolidating commercial workloads," *18th Large Installation System Administration Conference (LISA 2004)*, November, 2004.

[20] Pruyne, J. and M. Livny, "Managing checkpoints for parallel programs," *2nd Workshop on Job Scheduling Strategies for Parallel Processing (In Conjunction with IPPS '96)*, Honolulu, Hawaii, April, 1996.

[21] Rashid R. and G. Robertson, "Accent: A communication oriented network operating system kernel," *Proceedings of the 8th Symposium on Operating System Principles*, pp. 64-75, December, 1984.

[22] Rescorla, E., "Security holes... Who cares?" *Proceedings of the 12th USENIX Security Conference*, Washington, D. C., August, 2003.

[23] Rozier, M., V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrman, C. Kaiser, S. Langlois, P. Léonard, and W. Neuhauser, "Overview of the Chorus distributed operating system," *Workshop on Micro-Kernels and Other Kernel Architectures*, pp. 39-70, Seattle, WA, 1992.

[24] Saltzer, J. H., and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, Vol. 63, Num. 9, pp. 1278-1308, September, 1975.

[25] Sapuntzakis, C. P., R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, "Optimizing the migration of virtual computers," *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.

[26] Schmidt, B. K., *Supporting Ubiquitous Computing with Stateless Consoles and Computation Caches*, Ph.D. thesis, Computer Science Department, Stanford University, 2000.

[27] Smith, P. and N. C. Hutchinson, "Heterogeneous process migration: The Tui system," *Software – Practice and Experience*, Vol. 28, Num. 6, pp. 611-639, 1998.

[28] Swift, M. M., B. N. Bershad, and H. M. Levy, "Improving the reliability of commodity operating systems," *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 207-222, ACM Press, New York, NY, 2003.

[29] VMware, Inc., http://www.vmware.com .

[30] Whitaker, A., M. Shaw, and S. D. Gribble, "Scale and Performance in the Denali Isolation Kernel," *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, December, 2002.

Appendix

To isolate regular Linux processes within a pod, AutoPod interposes on a number of system calls. Below we provide a complete list of the small number of system calls that require more than plain virtualization. We give the reasoning for the interposition and what functionality was changed from the base system call. Most system calls do not require more than simple virtualization to ensure isolation because virtualization of the resources itself takes care of the isolation. For example, the kill system call can not signal a processes outside of a pod because the virtual private namespace will not map them and therefore it cannot reference it.

Host Only System Calls

1. mount – If a user within a regular pod is able to mount a file system, they could mount a file system with device nodes already present and thus would be able to access the underlying system directly in a manner that is not controlled by AutoPod. Therefore, regular pod processes are prevented from using this system call.
2. stime, adjtimex – These system call enable a privileged process to adjust the host's clock. If a user within a regular pod could call this system call they would cause a change on the host. Therefore regular pod processes are prevented from using this system call.
3. acct – This system call sets what file on the host BSD process accounting information should be written to. As this is host specific functionality, AutoPod prevents regular pod processes from using this system call.
4. swapon, swapoff – These system calls control swap space allocation. Since these system calls are host specific and have no use within a regular pod, AutoPod prevents regular pod processes from calling these system calls.
5. reboot – This system call can cause the system to reboot or change Ctrl-Alt-Delete functionality and therefore serves no place inside a regular pod. AutoPod therefore disallows regular pod processes from calling it.
6. ioperm, iopl – These system calls enable a privileged process to gain direct access to underlying hardware resources. Since regular pod processes do not access hardware directly, AutoPod prevents regular pod process from calling these system calls.

7. create_module, init_module, delete_module, query_module – These system calls are only related to inserting and removing kernel modules. As this is a host specific function, AutoPod prevents regular pod processes from calling these system calls.
8. sethostname, setdomainname – These system call sets the name for the underlying host. AutoPod wraps these system calls to save it as a pod specific name and allows each pod to call it independently.
9. nfsservctl – This system call can enable a privileged process inside a pod to change the host's internal NFS server. AutoPod therefore prevents a process within a regular pods from calling this system call.

Root Squashed System Calls

1. nice, setpriority, sched_setscheduler – These system calls lets a process change its priority. If a process is running as root (UID 0), it can increase its priority and freeze out other processes on the system. Therefore, AutoPod prevents any regular pod process from increasing its priority.
2. ioctl – This system call is a syscall demultiplexer that enables kernel device drivers and subsystems to add their own functions that can be called from user space. However, as functionality can be exposed that enables root to access the underlying host, all system call beyond a limited audited safe set are squashed to user nobody, similar to what NFS does.
3. setrlimit – this system call enables processes running as uid 0 to raise their resource limits beyond what was preset, thereby enabling them to disrupt other processes on the system by using too much resources. AutoPod therefore prevents regular pod processes from using this system call to increase the resources available to them.
4. mlock, mlockall – These system calls enable a privileged process to pin an arbitrary amount of memory, thereby enabling a pod process to lock all of available memory and starve all the other processes on the host. AutoPod therefore squashes a privileged processes to user nobody when it attempts to call this system call to treat it like a regular process.

Option Checked System Calls

1. mknod – This system call enables a privileged user to make special files, such pipes, sockets and devices as well as regular files. Since a privileged process needs to make use of such functionality, the system call cannot be disabled. However, if the process could create a device it be creating an access point to the underlying host system. Therefore when a regular pod process makes use of this system call, the options are checked to prevent it from creating a device special file, while allowing the other types through unimpeded.

About the Integration of Mac OS X Devices into a Centrally Managed UNIX Environment

Anton Schultschik – ETH, Zurich, Switzerland

ABSTRACT

The UNIX flavors in use today have so much in common that centralized management of UNIX systems has become almost standard. Since Mac OS X is based on BSD-UNIX it is a promising candidate for integration into a centrally managed UNIX environment.

Starting from generic administration concepts, this paper develops an integrated management concept that handles fully automated installation and configuration of hosts. The concept includes a centralized application management system for console and graphical Mac OS X applications.

The management concept is then implemented based exclusively on standard UNIX tools. The necessary extensions of these tools to make Mac OS X conform to UNIX standards are presented, including a proxy tool to forward AppleEvents which facilitate the interprocess communication for centrally managed graphical Mac OS X applications.

Introduction

The increasing diversity of hardware and software makes system management more difficult. Shorter life-cycles of computer systems require more frequent upgrades or replacement of hardware and as a consequence, the installed computers on a large site rarely are uniform in hardware but rather split into several uniform clusters. Automated management of such an environment is challenging as complexity grows with each new configuration of hardware and software.

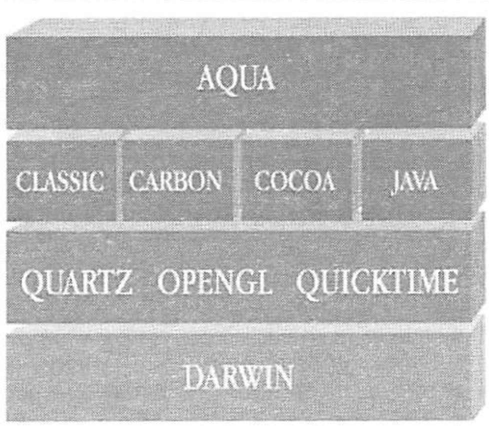


Figure 1: Structure of the Mac OS X operating system (from [2]).

Through integrated management of the clusters common tools and common configuration information can be reused across the clusters thus reducing the amount of information required to fully understand the entire site. With Mac OS X being a member of the UNIX family integrated management of Mac OS X in a UNIX environment comes within reach. In this paper, design, implementation and deployment of such a system will be layed out built on some basic principles. The result will be a management system for UNIX systems that can be used to manage network-based as well as stand-alone systems.

In system management three basic principles keep appearing in tools and methods [1], and these shall be used as orientation for our integrated management system:

- *Reproducibility* ensures that the same action produces identical results. Automation, a way to implement reproducibility, helps to exclude human error in repetitive tasks.
- *Comprehensibility* of all actions is necessary for the administrator especially when troubleshooting or modifying the configuration.
- *Avoidance of Redundancy* helps to keep data consistent and thus easier to manage.

Mac OS X is UNIX Plus . . .

Mac OS X unites the strengths in UI and application design of previous Macintosh operating systems with the stability and flexibility of a modern UNIX platform. Integrating Mac OS X into a general UNIX environment requires a closer look at the operating system since not all parts in Mac OS X have their origin in the UNIX world:

- **Darwin**: Darwin is based on Free-BSD including the standard UNIX network clients and servers as well as the usual user space utilities. The appearance of some daemons and configuration files have been modified to match with the rest of Mac OS X.
- **Quartz, OpenGL, QuickTime**: Instead of relying on the UNIX X11 standard, Apple

decided to build an alternative graphic system.

- **Classic**: The Classic environment provides emulation support for native pre-OSX applications. These applications only work with Apple's HFS/HFS+ filesystem.
- **Carbon**: The Carbon library framework provides compatibility to pre-OSX system calls at source code level. Carbonized applications also run on non-HFS filesystems through Carbon's HFS emulation although with reduced stability.
- **Cocoa**: Providing an entire new standard for application development, Cocoa is based on modern, UNIX compatible technologies like XML and Java.
- **Aqua**: The top layer of Figure 1 represents the graphical user interface on which the different GUI applications run.

Looking at the entire operating system, the Darwin roots as well as modern Cocoa-based applications are fully compatible with the rest of the UNIX world even though Apple did not use the X11 standards in their graphics system. Thus the key challenge in managing a Mac OS X system as a UNIX is the handling of legacy applications and their specialties.

Review of Available Tools

Several system management tools are available under Mac OS X that focus on three different management areas:

- **Installation** of the operating system
- **Configuration** of operating system and applications
- **Software distribution** or installation onto an installed host

The management tools need to be applicable on classic UNIX flavors while supporting the Mac OS X specific extensions, e.g., legacy application support. Several candidate tools were considered, and their strengths and weaknesses will be discussed in the following sections.

Installation Tools

Several strategies can be chosen to install an operating system onto a target host. Network-based installation allows access to centralized services and is logistically efficient. Therefore only network-based installer methods are considered in the choice of tools.

NetInstall

Apple provides a native mechanism [3] to install multiple client machines based on the installation of packages (.pkg bundles [4]). The target host of an installation is net-booted from a modified disk image that will start an installer system. The installer then installs packages supplied on the booted image.

The package-based approach of NetInstall yields reproducible and certainly comprehensible results since all changes on the installed system have their source in one of the installed packages. As the installer image can only contain a single configuration several NetInstall images are required when managing multiple host configurations. Consequently, each image would redundantly contain commonly installed packages making the management of NetInstall a difficult task in a heterogeneous environment.

Net-Restore

Mike Bombich's NetRestore [5] is a suite of GUI tools that are based on ASR, Apple's image management tool [6]. An installation is started by net-booting a target host into the NetRestore installer in which the administrator selects the image to be restored to the local disk. The individual images are supplied through a network share along with post-installation scripts for the individual configuration of the installed host.

Since the restored system is identical with the source image the installation itself clearly is reproducible. However creation and maintenance of the source image is done by hand and the final system can not be comprehensible as a whole. An administrator neither explicitly sees why a system is in the current state nor completely understands the consequences of each manual step during assembly. As with NetInstall in the previous section the management of several configurations implies the use of multiple disk images introducing redundancy between the manually maintained images.

Sun Solaris Jumpstart

Network based installation is done by net-booting an installation target into the Sun Solaris Jumpstart [7] system. Once booted Jumpstart uses DHCP and DNS to determine the correct configuration list of packages and appropriate pre/post-processing scripts.

The Jumpstart configuration concept is simple and yet capable of comprehensibly handling individual configurations. Its design for completely unattended installation makes the Jumpstart system reproducible.

Configuration Tools

To reproducibly maintain the configuration of systems, especially in a heterogeneous environment, automated tools are essential. However to provide the necessary comprehensibility, configuration information consisting of a large number of modifications for a target system must be structured into modules. By postulating module integrity, i.e., that no module destroys the modifications of another, reuse of modules becomes possible, thus controlling unwanted redundancy. The following three tools fulfill this basic requirement.

Cfengine

One of the best known tools comes from Mark Burgess of Oslo University College. Cfengine [8] is a highly flexible scripting system that deducts its configuration based on the context of a managed host. Cfengine supports various UNIX flavors including Mac OS X and is equipped with its own file sharing mechanism.

Provided that Cfengine is run in the same context, reproducible results can be expected, and modularization is provided through the *classes* construct.

However, since Cfengine does not enforce integrity of actions or classes, configuration scripts can easily exceed the state of comprehensibility.

Radmind

Radmind [9] is available on various UNIX dialects including Mac OS X. Designed for ease of use, Radmind implements a capture and replay strategy resulting in a tree of dependent *load sets* consisting of files modified during a capture session.

The capture and replay processes alone are comprehensible and reproducible. However the dependencies between load sets restrict the replay of features between different lines of history. As these restrictions are not managed by Radmind the replay of an unsuitable load set impairs the reproducibility of functionality and the comprehensibility of the configuration.

Template Tree 2

Template Tree 2(TeTre2) [10] is used for the administration of UNIX clusters. The configuration of a system is managed using simple file operations structured into integrity-preserving modules. A subset of Cfengine functionality is used to propagate the configuration to a target system.

Although dependencies between features in a TeTre2 configuration exist, these dependencies are functional rather than historical. This allows features to be recombined freely without impact on comprehensibility or reproducibility.

Software-Distribution Methods

When managing software, the integrity of application packages and the operating system ensure that all available applications are functioning. Knowing the origin of each file in the system is the basis to maintain such integrity. Especially when software packages are installed intrusively, i.e., by copying them into the standard directory tree of a system, the tracking of file origins becomes difficult and thus needs to be examined more closely.

Fink and the Debian Packaging System

Software built within the Fink Project [11] is packaged in Debian packages, that are installed with a well supported tool-suite. A database keeps track of the origin of installed files by associating them with their source package. An installer then relies on this database to ensure the package integrity.

Software management using Debian packages is thus reproducible and comprehensible. In a larger environment where multiple versions of the same package need to exist file collisions can only be resolved through renaming or versioning, e.g., `gcc-2.95` and `gcc-3.3`. As not all files are easily relocatable and the resulting versioned structure always enforces version dependencies, creation and administration of such packages would be expensive.

Network-based Distribution Via SEPP

The SEPP-Packaging [12] system follows a different concept to distribute applications. Rather than installing applications file by file into an existing installation each *SEPP-package* encapsulates a ready-to-use application within a separate directory. Once this package directory has been copied or mounted over NFS the packaged applications are made accessible to the user using stub scripts. By design SEPP supports the coexistence of multiple versions even when dependencies on other SEPP-packages exist.

Through its structure a SEPP installation is comprehensible and yields reproducible results. In addition the Mac OS X standard proposes a similar approach to encapsulate the files of an application into a bundle directory. Thus incorporating Mac OS X support into SEPP is straight forward.

A System Management Concept for Mac OS X

Available Infrastructure

When implementing our Mac OS X management concept, we relied on the existing infrastructure at our site. This infrastructure consists of the provided network services and the available tools. These two aspects of the infrastructure are discussed in this section.

Available UNIX Network Services

Centralization of services is a common approach to reduce redundancy of information. In the implementation of the integration concept the following services were used:

- **DHCP**: Management of network configuration and access control
- **LDAP**: Consistent user authentication and group management
- **NFSv3**: Network-based user homes and application packages
- **SMB**: External access to user homes

While all other services are only available using a single protocol to prevent confusion, user homes can be accessed using two different services: NFS and SMB. NFS is used as default for reasons of speed and flexibility. However the current implementation of NFS requires a trusted network and accordingly SMB is used for remote access from home or across untrusted networks.

Chosen Tools

Several administration tools were introduced earlier, each being particularly strong in one of the administration areas. This information is now used to determine the best suitable tools to implement an integrated concept:

- **Network Installation of Mac OS X**: Jumpstart is the best choice even though reimplementation is necessary because it runs fully automatic and completely unattended.
- **Configuring the System**: The built-in limitation of complexity as well as the support for modularization make TeTre2 the preferred tool for a structured approach to integration.

- **Serving Applications**: Because the SEPP approach matches with the Mac OS X application concept SEPP is an ideal choice even though an extension for graphical applications needs to be implemented.

In the next sections we first introduce each of the tools in its original form followed by a description of the changes necessary to make it work on Mac OS X.

Network Installation of Mac OS X Based On UNIX Standard Protocols

Design of Solaris Jumpstart

Jumpstart is based on the standard UNIX services BOOTP/DHCP, TFTP and NFS. Once it is net-booted, it performs an unattended installation as described in Figure 6.

The jumpstart system is net-booted on the target host. Therefore the DNS name of the target host is known to jumpstart at execution time and thus can be used to access a configuration directory on an NFS share with the name of the target host. This directory contains the pre-/post-installation scripts and a list of packages that shall be installed.

The Reimplementation of Jumpstart for Mac OS X

Our Mac OS X reimplementation of Solaris Jumpstart provides the same basic functionality and several useful extensions. Some of these extensions are listed in gray color in Figure 6.

- **DHCP**: Although Darwin is compatible with UNIX standards, the net-boot implementation of Macintosh open-firmware is incompatible with the normal DHCP protocol and tries to start negotiations with the DHCP server. With a patch found in [13] it was possible to net-boot the installer using Linux and Sun Solaris.
- **Override for hostname**: One configuration file in TeTre2 associates all host names with their Ethernet MAC addresses. With the information from this file osxjumpstart can set the correct host name even when a host is booted of a temporary IP address and choose a different host name for the configuration. We use this feature when installing portable Macs from a common set of "install only" IP addresses.
- **OS Installation**: Since applications are supplied using SEPP, only the operating system and extensions, e.g., fonts, are installed via osxjumpstart. The OS packages are provided in their native .pkg bundle form [4, 14]. Additional self-built packages are also provided in this format for consistency.
- **Software Update**: Using Apple's softwareupdate tool Mac OS X and installed packages are brought up-to-date. Unfortunately softwareupdate was designed to update an already running system. Thus the tool is run in a chrooted environment on the newly installed system without requiring a reboot.
- **Management of Sensitive Data**: All sensitive data, e.g., passwords, licenses, . . ., is stored in an encrypted archive and the administrator is required to enter the encryption key at the beginning of the installation. In the post-install actions sensitive data is then copied into the system.

  Passwords for users in the netinfo database are stored in separate files under /var/db/shadow/hash. The hash files can be copied to the archive from any Mac OS X system with appropriately set passwords. The same strategy can also be applied to the Auto-login password (/etc/kcpassword) and the Open Firmware password (nvram security-password and nvram security-mode).

Configuring the System

Design of Template Tree 2

Template Tree 2 (TeTre2) is a configuration management system that structures the configuration of an entire site into modules – *features* in TeTre2 terminology. The configuration of a host is defined as a set of features whereas features consist of simple file operations. Figure 7 shows the structure of TeTre2. Basic information like hardware type or Ethernet MAC

Net-Boot	• Determine IP using DHCP or BOOTP • Boot the installer system from TFTP, NFS • Mount required NFS shares
Config-retrieval	• Get host name from IP-number and DNS • Override host name for specified Ethernet address • Use hostname to retrieve config info from NFS
Pre-Install	• Format/partition disks
OS/SW installation	• Install required packages • Update the system using softwareupdate
Post-Install	• Configure the system (TeTre2) • Transfer sensitive data • Optional custom actions
Reboot	• Start the installed system

Figure 6: Steps of a jumpstart installation (black) and the extensions for osxjumpstart (gray).

address of each host (the host.list file) on a site is separated from the individual host configuration for all managed hosts (file site.desc).

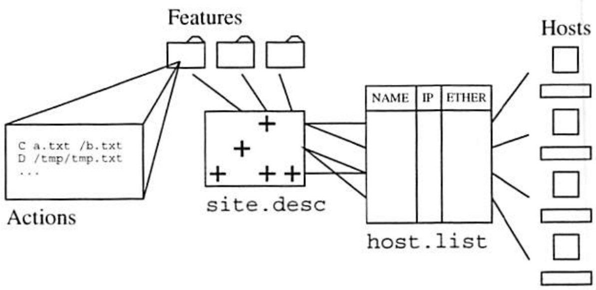


Figure 7: Structure of the TeTre2 Configuration: In the host.list a hostname is associated with basic configuration information like installed OS, Ethernet Address or Network configuration that can be used as text substitutes in configuration actions. The site.desc file then selects the features for a host. Finally each feature contains a file containing a list of configuration actions that are applied to a host if the associated feature has been selected in site.desc.

Features encapsulate a certain functionality, e.g., install and activate a service, or behavior, e.g., power management. The ability to enforce integrity allows flexible combination between features and avoids coincidental dependencies. Dependencies between features are designed by the administrator. As a consequence TeTre2 features obtain a semantic aspect in the context of system administration.

Looking inside a feature one will find simple file operations such as copying files, creating and removing directories and generating symlinks. The assembly of files from chunks, simple text substitution and operating system dependent execution of file operations provide more flexibility for the configuration work. For the sake of comprehensibility TeTre2 never modifies the content of files. Either the files are fully controlled and consequently overwritten or the files are outside the scope of TeTre2 and thus ignored. Files with mixed content, e.g., that partially depend on TeTre2 configuration are managed by custom scripts running on the target system. For example allowing users to add their own printers and still manage a default set of printers through TeTre2 requires this mixed form of control.

Overview Over the Mac OS X TeTre2 Features

The functionality of the TeTre2 tool itself was sufficient to configure Mac OS X systems without modification. However a multitude of new features were required to control the diverse aspects of a Mac OS X installation. Figure 8 gives an overview of the implemented Mac OS X features. The categorization of the features is not part of TeTre2 but was introduced in the table for the benefit of the reader.

Basics:	Automounter, Network configuration, Netinfo builder
Admin tools:	SSH public-key access, automatic package installer
Services:	Postfix, sshd, Filemaker/Meetingmaker server
Clients:	LDAP access, automount NFS homes, SEPP
User specific:	Default printers, custom loginwindow dialog

Figure 8: Categories of TeTre2 features with examples.

Unless documentation is available, features are created by identifying and verifying the differences that occur when settings are modified through the Mac OS X GUI. The concerned files are stored in the feature and copied during the configuration process. If several features contribute to the content of a file, chunk-wise assembly is the solution. However when the content of a file depends on the current state of the system, e.g., hardware, the files are generated through scripts and the configuration for these scripts as well as the installation of the scripts are handled with TeTre2. Some of the features we added to TeTre2 for Mac OS X support were:

- **Package Installer** Adapted from the corresponding UNIX feature an automatic post-install .pkg bundle installation is performed through a script. The package installer is run at each system start as StartupItem and is also run nightly by cron.

  Packages missing on the local system get installed based on a configuration directory. Upgrades of existing packages and required reboots are handled, the latter is done through the reboot command or by sending an AppleEvent to the Finder when a user is logged in.
- **Netinfo Database Builder** Netinfo data is stored in binary format making direct control through TeTre2 impossible. As the Netinfo database is required by Mac OS X for some applications parts of Netinfo are regenerated at each start using the Netinfo Builder. Netinfo Builder is a Perl script that compares the current state of the Database with a desired state in the form of a raw dump (the term *raw* might be imprecise since the dump is ASCII but in a general format). Netinfo Builder only changes the differing elements allowing a coexistence between managed and individual settings.

Serving Mac OS X Applications from NFS

Design of SEPP for Command-line Applications

The design goal behind the SEPP [12] application distribution system is the encapsulation of ready-to-run application packages into individual directories. As a consequence the application executables inside a SEPP package need to be *reconnected* to the OS and its user. SEPP is structured into two parts that are

separated physically in two directory trees as shown in Figure 9. The application packages are located under /usr/pack/.... Optional support for NFS automount is built into SEPP allowing application packages to be assembled from various sources.

The /usr/sepp/ directory is used to manage the applications on the local system making them available through the /bin/... directory. Rather than symlinking the executables from an application package, a Perl script – a stub – stands for the real application binary. Upon execution the Perl stub (in /usr/sepp/bin/) starts the real executable inside the package triggering the automount of the package containing the application. More precisely the Perl stub starts a package specific start script named start.pl (inside each /usr/pack/...) which then can properly set everything up before starting the application. The concentration of all administration information into one directory (/usr/sepp/) makes it possible to distribute this directory via NFS resulting in an application system that is fully centralized.

SEPP for Mac OS X

SEPP supports Darwin and X11 applications on Mac OS X in the unmodified form. In the case of Cocoa or Carbon applications the basic structure of Mac OS X applications [15] comes into play: Mac OS X applications are packaged into a single, structured .app bundle directory containing application data or additional libraries required by the application. Usually the files and directories within the bundle are addressed by relative paths allowing an .app bundle to be moved around on the system without re-installation.

Making the packaged applications available to the Finder is more of a challenge. A closer look at the anatomy of a .app bundle however leads to the following solution. The Finder only needs a part of the bundle files to display the application and associated documents. We use this fact in the Mac OS X extensions of SEPP by copying all the required files to form an independent miniature application – an *application stub* – in the /usr/sepp/macosx/ directory. In this application stub a small Perl script is used to start the real application. Figure 10 shows the structure of such an application stub directory. By making the contents of the /usr/sepp/macosx directory available as an item in the Finder's left hand navigation bar users can access all SEPP applications in a natural Mac-like way.

Special Application Support

All binaries in a SEPP package are started indirectly through the package specific start.pl script. One of the advantages of such a script is the ability to prepare the environment for an application. Mac OS X SEPP packages make regular use of start.pl since many applications do not follow the standard exactly.

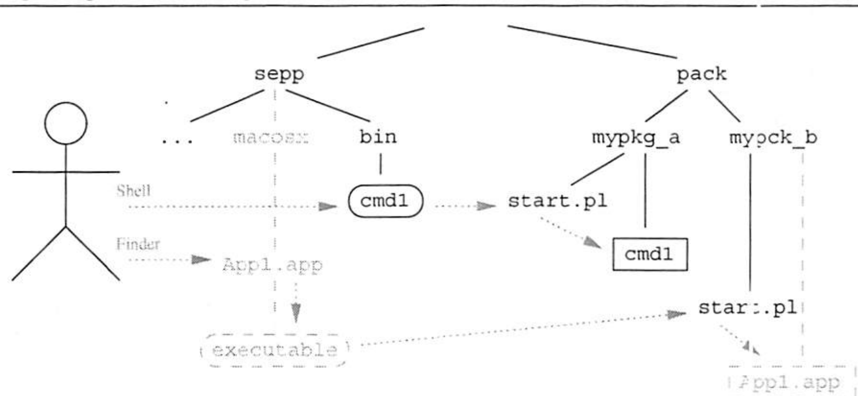


Figure 9: Structure of the SEPP system: The original SEPP system designed for command-line applications is shown in black and the extensions implemented to support Mac OS X GUI applications are shown in gray. The boxes in the figure represent executables. Rounded boxes are used for the Perl stubs that a user will execute directly while the cornered boxes stand for the real applications. The individual start.pl files found in every package provide customization for the application to be started. The dotted arrows show the chain of actions when an application is launched.

.../Sample.app/	
Contents/	
Info.plist	Contains filenames of executable and icons
MacOS/	
the_executable	Perl script replacing the real executable
Resources/	
sample_icon1.icns	Displayed in Finder
sample_icon2.icns	
....icns	All other Icons referenced in Info.plist

Figure 10: Structure of a Mini application bundle used as application stub. Entry point is the Info.plist file in which all other filenames of the bundle are found.

RegColl: Centralized Registry Framework for Infrastructure System Management

Brent ByungHoon Kang, Vikram Sharma, and Pratik Thanki
– University of North Carolina at Charlotte

ABSTRACT

System administrators are faced with the challenge of managing and compiling information about deployed systems to ensure the maintenance, scalability, security, and overall availability of the infrastructure systems. Recently, securing and monitoring the deployed enterprise infrastructure systems has taken unprecedented importance due to the added accountability now being placed on managing corporate data and private information. Compliance with new regulation has never been stressed more than now – with SOX/Bill 198, GLB Act/PIPDEA, HIPPA to name a few regulations.

Toward this, we present RegColl: A practical framework for registry and system configuration management for monitoring and maintaining the health of corporate infrastructure systems in Windows environment. In RegColl, registry and configuration information are collected at a centralized location so as to inspect and analyze the data for policy compliance, system configuration monitoring, and incident response services.

Introduction

Managing hardware, operating systems, and applications on workstations and desktops in corporate and large networked environments is a big challenge, one that System administrators deal with on a day to day basis. To monitor the health of infrastructure systems, it is imperative to understand system behavior, especially aspects that are related to the upkeep of the system and eventually the whole network. Windows *registry* provides system information to understand the behavior of the deployed applications.

Monitoring the system's registry and configuration information offers important insights in deployed software, patches, drivers, etc. – such entries can be analyzed for vulnerabilities, troubleshooting, and compliance with standards, along with providing documentation in an event that might have reporting implications. Further, by using the registry and administrative tools in Windows, system administrators can provide local or remote support, query registry logs for security analysis, and can apply compliance checks. We extend this facility in centralizing system monitoring services to run audits, apply compliance standard checks, and enforce corporate & legislature policies.

The problem of maintaining healthy configurations of a large installed base and other third party software packages has been recognized as a daunting task [10]. The growing numbers of software and changing system configuration makes it even more difficult to specify an "ideal configuration," which makes troubleshooting and maintenance problems intractable [8]. Aside from compliance and network monitoring issues, system troubleshooting expenses drive the total cost of network ownership, with maintenance and support costs. Largely, troubleshooting cases are due to system misconfiguration [8] and hence, it is one of the issues we propose to tackle with RegColl's centralized registry framework.

Non-compliance with legislature requirements such as Sarbanes-Oxley [16], Gramm-Leach-Bailey (GLB) Act [9], Health Insurance Portability and Accountability Act (HIPAA) of 1996 [11], and others that preserve private information is putting more and more organizations under scrutiny. Such compliance deviations are not only a breach of law but they also affect client confidence. Further, legislatures like OCC (Office of Comptroller of the Currency) [4] and HIPAA require organizations to maintain an information security program that includes "Detecting, Preventing and Responding to Attacks, Intrusions, or other Systems Failures."

Computer intrusions [5] leading to theft of intellectual property or any system compromise in a corporate network can jeopardize users' and employees' data integrity. Furthermore, any incident leading to misuse of confidential data can put a company's reputation at risk. Implication of such incidents may include comprehensive incident response plans and liability to notify the reporting agencies and the affected customers.

Traditionally management plans are devised for Network and IT Infrastructure which cover the security policies, audits, asset inventory, and incident response. Such plans ensure business continuity and disaster recovery in the event of computer and network intrusion or system compromise. Federal Trade Commission (FTC) Safeguards rule [7], OCC and Medical Privacy rules like HIPAA mandate an *incident response* plan which encompasses ways to confirm whether an incident occurred and provide accurate, relevant, and timely information.

We present the benefits of a centralized registry collection and monitoring approach in the next section. Subsequently, we give the architecture details of RegColl framework and its components including the collection mechanism and transfer mechanisms, and we discuss the implementation of RegColl framework. The related work section compares some of the existing technologies and their goals with RegColl framework. We then present the future work section, showing the ongoing development efforts and the proposed upgrades of RegColl framework. We then conclude, revisiting the usage and presenting the practical solutions that might be deployed to realize some of the key benefits of RegColl.

Why Registry Monitoring

To address requirements of system monitoring, configuration management, and incident response, we take a proactive system management. In this approach, we freeze configuration states that can stand comparison to a misconfigured system. In the event of an incident these frozen states provide a historic log of system registries containing restore points that allow for reconfiguration, System rebuild, and incident response.

The centralized registry offers key benefits like Policy enforcement at a centralized location, Audits and Analysis of configuration information, and the overall System Configuration Management. The following sections elaborate these benefits.

Policy Enforcement

Corporate and legislature requirements mandate formulation of a management plan and steps to control access to infrastructure systems, and ensure the security of financial data and non-public information. Centralized repository serves these purposes by providing a single point of policy validation and enforcement. Audits and compliance requirement checks are achieved by deploying compliance and auditing standards on the stored configuration.

- **Remediation** – Centralized configuration offers remediation capabilities to easily and effectively automate and monitor policy consistency requirements. The remediation technique allows for automated and user controlled repair process, for example, re-importing registries to enforce corporate policy across thousands of machines.
- **Restore Points and Time Travel** – Centralized configuration management allows for system restore and time travel by maintaining restore-histories. The causal relationship between the change information ensures a mechanism identifying the restore points and version changes. Such a tracking mechanism would enable reintegration of changes when a disconnected entity rejoins the network. Restore points and configuration change tracking features facilitate rollback to a stable state system state in the event of software misconfiguration.
- **Change Management** – Centralized configuration management detects and responds to planned and unplanned changes by setting standards and policies for change enforcement. Registry and configuration information captures the changes in system's configuration; any planned changes can be verified by studying the configuration change information.

Audit & Analysis

Risk management practices require audits and analysis of configuration information to assess and mitigate any risk arising from vulnerabilities in the system. Analysis of system, registry, and configuration logs is an important aspect of troubleshooting. Furthermore, such techniques take more prominence in rating and certification of processes and organization, and not ignoring that they are legislature requirements as well. Towards this, a centralized repository will serve as a common source for audit and analysis such as:

- **Security Auditing** – The RegColl framework proposes to collect and compare thousands of configuration elements from a networked system to identify and enforce security configurations standards. For example, security relevancy check [18] is done by inspecting security relevant registry keys.
- **Dynamic Auditing** – Centralizing registry and system configurations for *dynamic auditing* allows for systems to be audited seamlessly at a central and secured location on a regular basis. Auditing ensures consistent compliance and provides detailed reporting to address regulatory compliance requirements.
- **Security Analysis** – Centralizing registries give a single source of security and vulnerability analysis for tools developed and discussed by Wang [8] & Wenliang Du [18].
- **Gap Analysis** – These stores prove to be a single source for running gap analysis along with assessment of system configurations for known configuration vulnerabilities including patch inconsistencies.

System Configuration Management

Registries and configuration information is shared and accessed by multiple applications, such as drivers, software, and patches – it is imperative to cache a snapshot of this information so as to establish checkpoints at every change. Registry changes are monitored and pushed to a centralized location as and when new changes are recorded. The monitoring process polls the registry and configuration files and triggers the export of registry database as key entries are changed in the registry. These system-state snapshots are stored into a centralized secure location to analyze system vulnerabilities, run audits, enforce corporate and compliance policy checks, and monitor for any rogue software installations.

Incident Response

Incident response identifies ways to confirm whether an incident occurred and then provide accurate, relevant, and timely information. In such an event, the incident information gives implementation controls to secure the systems and crime scene to protect individual rights established by policy and law. Towards such requirements, a centralized registry and configuration repository provides a clean backup of all registry and configuration changes as they happen.

Corporate Data Protection

Protecting corporate data such as customer information is part of regulations, including the statewide California Database Security Breach Act, the Gramm-Leach-Bliley Act for financial services firms, and HIPAA for health care providers. These regulations are factors that drive corporations to beef up efforts to prevent unauthorized disclosure of sensitive data. In the recent past, several corporations, including large banks (Bank of America and Wachovia), have acknowledged substantial data leaks affecting tens of thousands of customers. Evidently, such leaks originate from inside the corporate office itself, often with data being transferred to or from a range of peripherals such as zip drives, CD/DVD drives and portable printers, to name a few. Furthermore, monitoring system ports and other peripheral devices can be used to block the entrance of viruses and mal-ware, and prevent corporate data from being copied or saved except where and when authorized. RegColl runs on all nodes on all the network clients and interrogates the Registry of every system attached to the corporate LAN. RegColl analyzes the registry's record of activity to identify what interfaces are being used, what peripherals are being attached to the system, and which of those devices are currently active.

RegColl's ability to examine peripheral devices connected to system allows for enforcement of data protection policies. For example, defining a corporate data protection policy, which makes it illegal to copy or move data to portable devices, can be tracked by running the policy enforcement tools. RegColl is not designed to monitor and capture use of peripheral devices; however, it can extend the functionality to incorporate the configuration changes from hardware driver installations.

The next section delineates architectural components of RegColl's framework.

Architecture

The managed entities in this system are workstations, desktop computers, and laptop systems. A process, called RegistryMonitor, runs on the managed entities to collect the registry and configuration information changes. The following sections give the details of each of the components and their interactions. We introduce the version and causal relationship mechanism of the change sets, the collection points, and the collection server components.

Architecture Overview

The RegColl framework introduces a registry and configuration information dissemination mechanism supported with a centralized repository. To ensure better availability of this information and protect against a single point of failure, RegColl introduces a secondary fail over server, which is accessible to the monitor process in the event of primary server failure.

RegColl takes into account the disconnected operations. We assume that some managed entities may enter disconnected mode when mobile. In such an event the registry monitor process collates the information locally with a causal version log. This local storage ensures that no critical changes are disregarded or missed when disconnected. Since RegColl works with causally related changes, the registry and configuration changes are reintegrated with the previously collected versions.

Since RegColl's framework collects and stores system information in a chronological manner, this archive reflects restore points that can be retrieved based on a version identifier. The causality allows for time travel and undo capabilities. Maintaining such a version relationship ensures that a disconnected entity when joining the network can seamlessly integrate the collated change information with the repository. The following subsection provides a systemic view of the RegColl's components.

Collection Points

The registry and system configuration information is collected by the *registry-monitor* process deployed as part of the registry collection framework. The registry-monitor acts as a collection source on each of the infrastructure systems' managed entity to gather registry and system configuration changes. These updates are cached when the registry keys are modified or a new sub-key is inserted or when a new node is created. The centralized framework identifies the keys that are most often updated and reflect critical modifications.

Registry and system configuration files tend to grow with time and configuration changes, such as updating new patches for operating system, security updates and driver installation. These registry logs might swell from a few kilobytes in size to multi megabytes (between 20 KB and 600 MB) as time elapses and configuration changes are made. To overcome the problem of caching and transporting heavy files, RegColl uses a *differencer* to identify changed entries and transport these deltas [1] to a registry collection server.

Often computer systems in a network, especially those in corporate networks, may be disconnected for mobility. Such a scenario mandates local caching of registry change snapshots (referred to as change-sets

in RegColl framework). To prevent the collection point of a disconnected system to make failed attempts to contact the collection server, the cache is polled before each connection attempt to the collection server. After the system is reconnected, any pending change-sets are pushed before a new snapshot is generated (Figure 1).

In order to actively observe the registry changes, monitors that have an operating system hook can provide most efficient cues on the registry changes. RegColl incorporates Microsoft's FileSystemWatcher component to notify (callback) Collection Server in the event of any registry file changes. Since registry information is critical, it is by default authenticated and encrypted using symmetric key cipher and hosted in the registry store.

Collection Server

The registry collector (primary server) acts as a collection point for registry and system configuration file entries. These entries are archived in the history server with version information, and time stamped for chronological ordering. As mentioned, the version and ordering information gives undo and time travel capabilities, since causally related entries can be queried for a particular version. The Collection Server archives the restore point for each system and maintains a mapping between the monitored system ID and the collected configuration files. The fail-over server architecture offers robustness and reliability to the overall framework. Exported registry entries are hosted on the primary registry server; in an event of server outage, the secondary server acts as backup. The primary and secondary servers synchronize maintaining the consistency in registry updates. This data consistency and synchronization is controlled using a causal history based replication framework [3].

Configuration and Registry Repository

Configuration and Registry repository provides a uniform monitoring infrastructure and is the single source of information necessary to develop the best practices for change, problem, and incident management. This source of archived registry & configuration log helps in configuration discovery & dependency mapping, and change tracking serves as the trusted information base for IT operations improvement.

The principal goal of configuration and registry repository is to tag the change-set version and store them in a multi-mapped manner while preserving their chronological order. This repository provides an accurate, always up-to-date configuration repository of the servers, infrastructure software, and dependencies between registries. Centralized registry enables system monitoring services and is a definitive source of automated, application and infrastructure configuration information.

Compliance and Analysis

A registry repository's ultimate goal is to serve as a single point of network monitoring service. The constructive contribution of a centralized registry and configuration store would depend on what tools are employed to query the repository and how frequently the repository is checked for compliance. The frequency of compliance check for policy compliance determines the effectiveness of centralizing the registry.

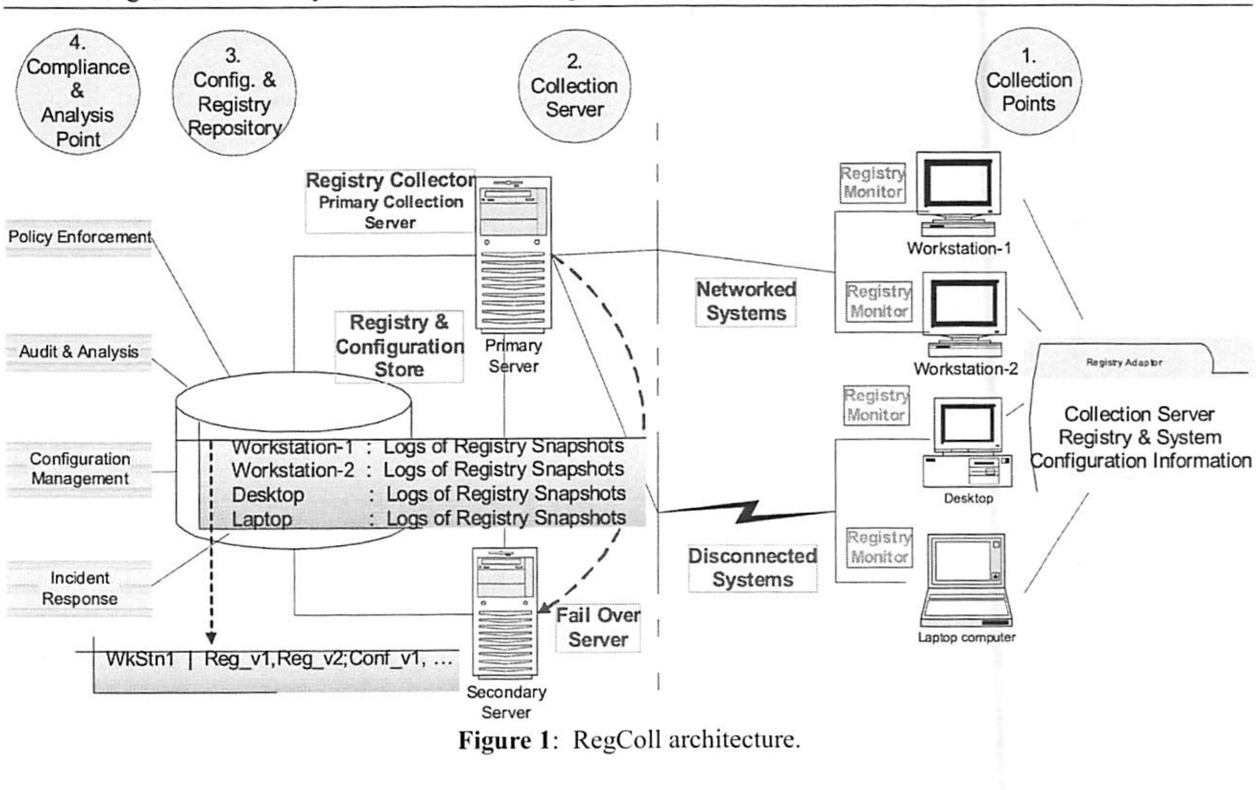


Figure 1: RegColl architecture.

Compliance checks at regular intervals would be able to respond to compliance deviation more effectively. However, a compliance check in the passive mode may be constantly looking for a pattern of malicious behavior. For example, if a policy deviation or deficiency is found, the system may trigger a host of alerts to isolate the host immediately until the host has been remediated. A host of third party software [17] can be employed to check the legislature requirements and perform system analysis.

Security Consideration in RegColl

The centralized registry framework requires entities to establish their identities in the form of public and private key pair. The creation of public-private key pair for each machine is performed as a system registration step. Registry logs that are collected as change sets (at collection points) are signed by the system/node's private key, which are authenticated to enable uploading to the collection server. Signing change sets between the collection-point (Registry Monitors) and the collection server (Registry Collector) establishes confidentiality and proper usage.

The framework enables trust in collection entities by certificate exchange. Entities are governed by their identity keys (private keys); hence, registry information shared is secured from external attacks and malicious updates. Such a trust-framework enables revoking a key in the event of a malicious change-set being generated.

Another important security aspect is the user's system privacy and invasiveness of remote monitoring tools. We contend that our framework is less invasive, since the differencer tool compares and generates deltas for each new registry change. Change set deltas enables RegColl's registry monitoring service to consume orders of magnitude less bandwidth than any other active monitoring process. Further, remote management and monitoring tools (like, Tivoli and OpenView) allow for exclusive control of the system, wherein RegColl framework's registry snapshots always has read access only. The system administrator will never have access to any other file, hence preserving user's privacy.

Registry and Configuration Change Monitoring

We explain the registry and configuration change-set generation mechanism in this section. The change capturing process requires a policy to determine the frequency of changes. We share our experience in determining these policies along with a discussion on Operating System tools used for capturing registry and configuration changes.

Registry change generation synchronizes the local registry changes of managed entities with the remote collection server. These managed entities host a registry monitoring tool called RegistryMonitor, which publishes the change-data from the managed entities based on an "auto-publish" policy.

The *publish operation* policy determines how often the registry changes should be published or what should trigger the synchronization. In RegColl's implementation we reviewed publish policies based on "elapsed time" and "transaction size" of the registry updates. Elapsed time mechanism assumes changes to be comparatively smaller if the time units are maintained small enough. This proved to work fine until the user tries to install large software. For example, installation of Oracle database induces numerous registry changes. We found that large installation generates so much traffic that it affects the efficiency of change-set generation process. Hence, we recommend fine-tuning the change-set generation and publish policies in the event of planned large installations.

In the implementation scenario, another challenge is determining the publish policy. The policy should be set for smaller transaction size or smaller time interval, which inversely affects the change-set size. Too fine granular publishing affects the performance and usage of the system. On the other hand too coarse grained or too many updates are found to produce very large change-set, which are detrimental in exchange operation and hence, proves to be an inefficient use of server resources.

Our experience in determining the publish policy comes from monitoring native file system changes using FileSystemWatcher component on Windows [13] and Inotify file system event monitoring mechanism [6] for Linux. FileSystemWatcher, of Microsoft .NET framework's System.IO namespace gives access to system functionalities such as the FileWatcher utility. The FileWatcher listens to the file system change notifications and raises events when a directory or file in a directory changes. This facility is utilized to monitor registry-file changes.

Differentiating between users initiated change and OS process changes are not discernible to the file watcher. For example, interaction between Windows Explorer (explorer.exe) and user workspace are reported as file changes. Further, any file creation leads to multiple file change notification.

On a Linux based system, a number of swap files are created when user programs open a file. Our experience shows that these files are created and deleted within a few milliseconds. Capturing these temporary files as part of change-set description proves to be unwanted overhead, both in terms of network traffic and change set log size. To manage such copious change notifications, we analyzed both FileWatcher and Inotify change log pattern and developed a filter routine that prunes junk and redundant entries.

RegColl follows a transactional semantics where all changes take place atomically. A time based publish policy will divide the transaction into multiple units should the installation triggered change time is greater

than that of publish time. For example, software like Tomcat web server has a longer installation and configuration time than the default publish time. We ran our experiments on University network configured to trigger notification in the events of file creation, file access and write, file name and directory name change. The size and security attributes changes are also reported. We found that the file system watcher component has certain limitations, especially in reporting what process initiated the registry-file changes.

We contend that monitoring only the registry-file changes will improve the overall performance of synchronizing the configuration files since only the registry changes are propagated to the server reducing the overall network traffic. Even with snapshotting technique, the snapshot size can be prohibitively expensive given that changes happen in number of configuration files; on the other hand, RegColl captures specific registry and configuration changes in a given time window.

Based on our experience in analyzing change notifications, we devised the staging policy where file change notifications are collected for 30 seconds and flushed in an event if no new change is detected in this time window, i.e., the inactivity window. An alternative scheme for pushing the change-set is if the queue reaches a predetermined size (e.g., 100 numbers of changes in our case). Nonetheless the publish policy is configurable.

RegColl Implementation

The Windows registry is a hierarchical structure of key and name value pairs. This structure acts as a central configuration database for users, applications, and other system information. Each key in the registry is a node in the hierarchical structure with one or more sub keys associated with it. This structure binds together configuration information in the name-value pair container associated with each node.

A "registry key" is of importance if it can reflect on configuration changes. If such keys and other system configuration information are collected at one location, it provides a centralized source of registry and configuration information that can be used for system auditing and analysis. To realize this single source of system information repository the following section gives the implementation details of the proposed framework.

Registry Monitor

The centralized registry collection framework requires network wide deployment. The current implementation bundles the collection point interface as Registry monitor batch file, regmon.jar, and regmon.properties files. RegistryMonitor batch file runs the Registry Monitor on the user's local workstation; regmon.jar has the APIs to identify registry changes

```
C:\WINDOWS\system32\cmd.exe

C:\regcoll\src>regmonitor

   Usage:  regmonitor [-options]

        where options include:

        -login <usr> <pwd>          authenticates user on collection server
        -logout                     invalidates session and logs out
        -start                      starts registry monitor
        -monitor                    monitors default registry and system configuration file
        -monitorpath <file path>    monitors configuration file specified
        -monitorall  <file path>    monitors default registry and specified configuration file
        -showlog                    show registry change log
        -verbose                    enable verbose mode
        -server                     show current collection server session
        -help                       show usage
        -exit | -quit               exit system
```

Figure 2: Registry Monitor interface options.

```
C:\WINDOWS\system32\cmd.exe

C:\regcoll\src>regmonitor -monitor

-------------------------------------
Registry files being monitored are:
-------------------------------------
C:\WINDOWS\system32\config\DEFAULT
C:\WINDOWS\system32\config\SAM
C:\WINDOWS\system32\config\SECURITY
C:\WINDOWS\system32\config\SOFTWARE
C:\WINDOWS\system32\config\SYSTEM
C:\Documents and Settings\pgthanki\NTUSER.DAT
-------------------------------------
```

Figure 3: Registry Monitor with "-monitor" command.

and communication protocol implementation for collection-point and collection server communication for pushing changes. The regmon.properties file has collection server information, i.e., collection server's fully qualified domain name (FQDN) and port on which the server is listening.

To fully understand the collection framework, we deployed and tested the system on a local Windows network. The following section gives a deployment scenario of registry monitor and registry collector.

Figure 2 shows the screen shot of Registry Monitor implementation. RegistryMonitor monitors the default registry files; DEFAULT, SAM, SECURITY, SOFTWARE, SYSTEM stored in the windows\system32\config directory and NTUSER.DAT stored in the *user home* directory. Files being monitored could be seen by invoking the "monitor" command, as shown in Figure 3.

Installation of any new application on Windows operating system essentially modifies four registry keys, i.e., HKCR (Classes Root Handle Key), HKLM (Local Machine Handle Key), and HKU (Users Handle key).

We tested Registry monitor by installing and uninstalling a few software and drivers. For example, as shown in Figure 5 and Figure 6, Registry monitor when started with the "-start" command has successfully identified a change in NTUSER.DAT registry file.

Figure 6 shows the registry modification after the user installed the 'Mozilla FireFox' application. In the event of Registry monitor polling the registry, the FireFox application installation will be identified as a registry change. Registry monitor's file differencer logs the change in the change set as shown in Figure 6.

Registry Monitor Setup

Registry monitor setup entails setting up a monitor code at each of the infrastructure system (what we refer to as, collection point). Each deployment requires setting up of public and private key pairs at the collection point. These keys secure the communication channel between collection points and collection servers, all registry change sets are signed with collection point's private key. The registration process requires the system administrator to authenticate themselves before enabling deployment process. Users set up a secured password for client log on process.

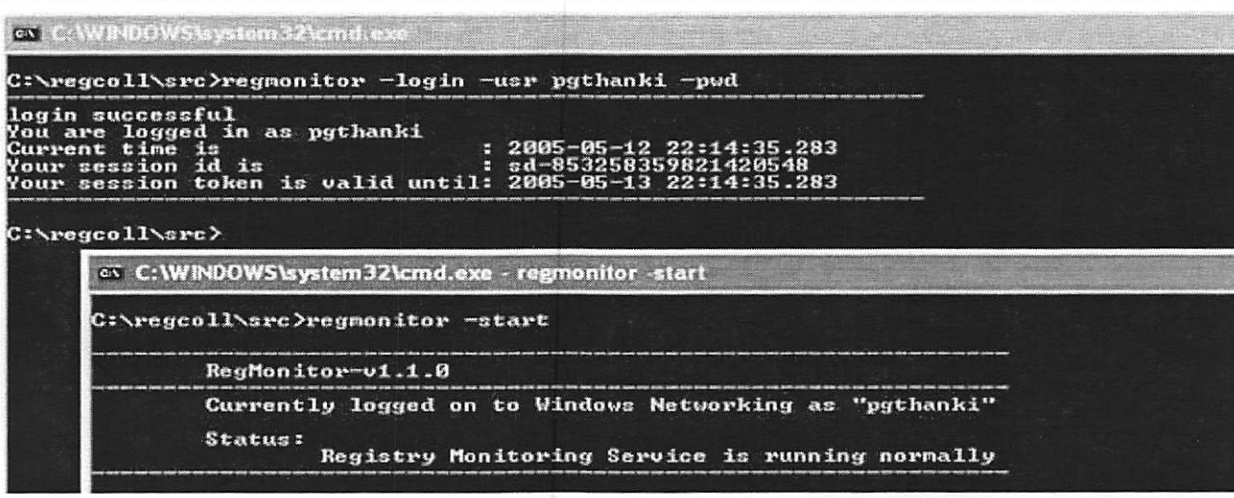


Figure 4: Registry Monitor when invoked with a "-start" command.

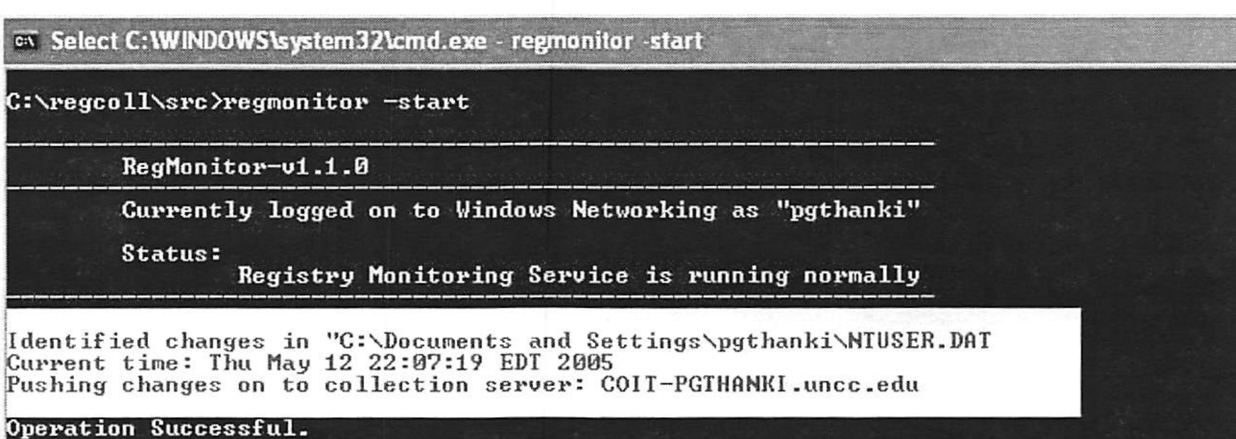


Figure 5: Identified Registry Changes using Registry Monitor.

Registry monitor requires users to authenticate with the collection server before initiating the registry monitor process. The logon process is session bound (Figure 4) and each login returns a session token from the collection server, which is usually valid for a period of 24 hours. The RegistryMonitor interface manages the session ID locally by encrypting it using a pre-established symmetric encryption key (set up during the machine registration process). User can logout of the system by calling "-logout" to invalidate the session token. The session token and session IDs are discarded once invalidated.

Registry Collector

The registry collector server maintains a directory structure of registry logs for each collection point (Figure 7). The registry collector captures the change-set snapshots, which are incremental delta information supplied by the collection point to reflect the registry changes. This contains the size of exchanged data at a given time since only the delta of change-sets are moved and not the whole registry or system configuration file. Moving an entire registry snapshot (taken at the time of registry change) is a process and network intensive operation. Hence, to avoid loading the network or system resources, only the change set deltas are generated and uploaded to the collection server.

These deltas give an incremental version change to the registry collection. Since the information traveling on the network is a delta of change set (or simply the changed data blocks), the information exchange is lighter and faster.

The registry collector maintains the version ordering for each change-set supplied by the collection points. These entries are multi mapped with MachineID, collection system's ID and the timestamp they are associated with. Figure-7 shows the collection servers format of storing change-set deltas.

Related Work

Monitoring system configuration information has been employed in many systems. The following are some examples. While these tools share the same idea of utilizing system configuration information as RegColl, they differ in the collection mechanism, such as disconnected mode support and its usages. RegColl collates the system configuration changes into a central location so that the centralized system configuration (e.g., registry) repository could incorporate the useful infrastructure system monitoring services such as compliance checks, incident response endorsement, and corporate policy validation. Similarly, Windows XP system restore [12] can collect registry data in its state snapshots.

The STRIDER [20] project uses differencing of periodic snapshots to reveal any configuration changes in the Windows registry. The registry keys of a failing program are monitored and recorded for analysis, this helps examine and reflect on fault in the program.

UNIX-based tools like Chronus [2] detect configuration error that might induce a faulty state in the system. Chronus captures the failure, inducing state changes to differentiate between working and non-working states. By using binary search, it can diagnose a range of common configuration errors for both client-side and server-side applications. It helps to reveal the specific failure cause, enabling recovery with minimal lost state. Another UNIX-based tool for system change monitoring is discussed in Backtracker [15]. Backtracker uses a change log mechanism and maintains an operating system causal history log. These logs are analyzed to determine the configuration changes which might be caused by the installation of an application or a computer intrusion.

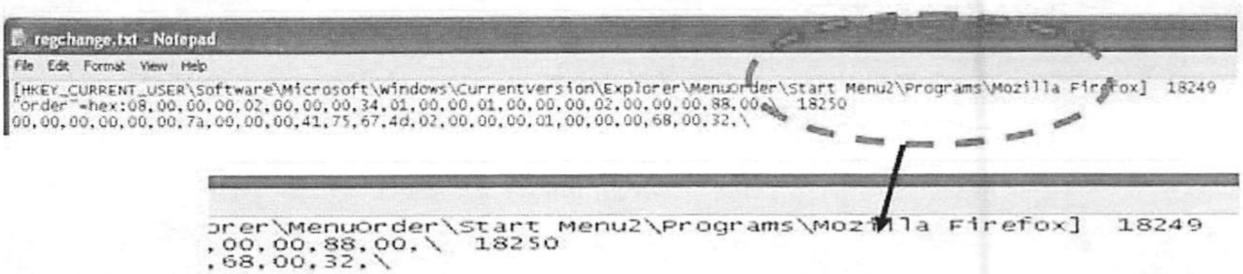


Figure 6: Changes identified by Registry Monitor.

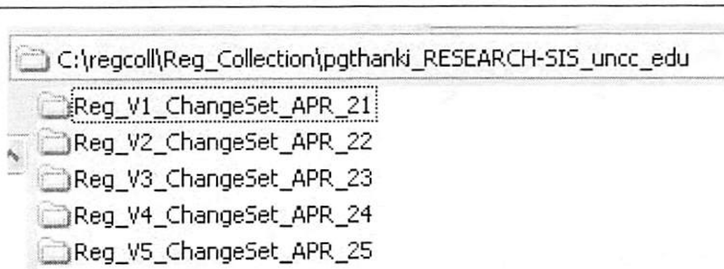


Figure 7: Directory structure of registry collection.

Future Work

RegColl is evolving rapidly and the skeleton is in place. We would like to invite users to try this tool, which can be extended to collect other system info files such as the .ini files on Windows and .rp resource files on UNIX systems. The tool is ready for download; please send an email to bbkang@uncc.edu for registry monitor bundle and/or any further information. This bundle, along with other resources can also be downloaded from the following link: http://coitweb.uncc.edu/~bbkang/ISR/.

Conclusion

The RegColl framework is positioned as a back end collection entity that seamlessly collates the registry and system configuration changes. A fail over server adds reliability to the architecture and provides a backup for change information. Deployment of such a server would obviously be on the network but away from the internet. Further, configuration servers are patently isolated and kept secure. However, we employ security considerations that mandate setting up public & private key pair at each collection point. All change sets are signed by the machines private key.

We delineate some of the useful contributions of system monitoring services that utilize the RegColl framework.

Corporate Policy Enforcement

The collection server can be deemed as a policy enforcement point, where policies are monitored and enforced by third party tools. This central location gives a validation source for such tools; if there is a policy deviation, a policy enforcer tool will generate an alert to the system admin. Audit and configuration are just logging tools, and a configuration management tool will identify if the registry is in good shape based on which rules the system administrator decides to apply. For example, Wang, et al, proposed a tool for troubleshooting misconfigured systems [8] using registry information.

Incident Response

Centralizing registry and configuration information meets the preliminary incident response requirements of documenting and confirming an incident. If an intrusion or system compromise is detected, it will be useful to analyze the changes that happened in the course of intrusion. The incremental change information is constructive in pinpointing specific changes. A compromised system's registry and system configuration information can be compared with a previously identified valid state.

Non-Invasive Monitoring

Active scans of a system's configuration and registry information can be considered more invasive process than capturing registry snapshots. The CPU and network usage of an active monitoring system (e.g., IBM Tivoli) consume order of magnitude more resources than a prescheduled maintenance task or remote monitoring.

Capturing registry information on a timely basis makes registry monitoring a lot less invasive than remote management, since the "differencer" produces deltas for each new change in the registry. Registry Monitor's monitoring service consumes orders of magnitude lower bandwidth than an active monitoring process. Remote management allows full control of the system where system configuration changes are accessible for read/write operations remotely. In comparison, the RegColl framework's registry snapshots are read-only. The collection server's administrator never has access to any other files, thereby preserving the user's privacy. The system configuration information is stored in an encrypted form which adds to the overall information protection.

The single source for auditing, analysis, and policy validations capabilities gives more control and monitoring power to system administrators managing large corporate networks and infrastructure systems. Hence, we believe RegColl's centralized registry and configuration framework will be a useful tool in overall infrastructure systems management.

Acknowledgement

The authors would like to thank the following people for their contributions to improve this paper: Gautam Singaraju of Department of Software and Information System, UNC Charlotte, and Vinod Eligeti of Department of Computer Science, Virginia Tech. We take this opportunity to especially acknowledge our shepherd, Yi-Min Wang of Microsoft Research, for his invaluable support in improving this paper.

References

[1] Tridgell, A. and P. Macherras, *The Rsync Algorithm,* Technical report, TR-CS-96-05, Australian National University, http://samba.anu.edu.au/rsync/ , June, 1996.

[2] Whitaker, Andrew, Richard S. Cox, and Steven D. Gribble, "Configuration Debugging as Search: Finding the Needle in the Haystack," *Proceedings of the USENIX Association OSDI '04: 6th Symposium on Operating Systems Design and Implementation*, 2004.

[3] Kang, B., Ph.D. Dissertation, *S2D2: A Framework for Scalable and Secure Optimistic Replication*, UC Berkeley, also in TechReport UCB//CSD-04-1351.

[4] Community Development Resource, Office of the Comptroller of the Currency Administrator of National Banks, http://citeseer.ist.psu.edu/377475.html .

[5] Computer Crime and Intellectual Property Section, http://www.usdoj.gov/criminal/cybercrime/cccases.html .

[6] Dow, Eli M., *Monitor Linux file system events with inotify*, IBM Linux Test and Integration Center, http://www-28.ibm.com/developerworks/linux/library/l-inotify.html?ca=dgr-lnxw07Inotify , 2005.

[7] The Federal Trade Commission (FTC) Safeguards Rule, *Financial Institutions and Customer Data: Complying with the Safeguards Rule*, http://www.ftc.gov/bcp/conline/pubs/buspubs/safeguards.htm , September, 2002.

[8] Wang, Helen J., John C. Platt, Yu Chen, Ruyun Zhang, and Yi-MinWang, *Automatic Misconfiguration Troubleshooting with PeerPressure*, Microsoft Research.

[9] "In Brief: The Financial Privacy Requirements of the Gramm-Leach-Bliley Act," *Federal Trade Commission – Facts for Business*, Available http://www.ftc.gov /bcp/online/pubs .

[10] Larsson, Magnus and Ivica "Crnkovic, Configuration Management for Component-based Systems," *Software Configuration Management – SCM 10*, 23rd ICSE, http://www.mrtc.mdh.se/index.phtml?choice=publications&id=0295 , May, 2001.

[11] Mercuri, R. T., "The HIPAA-Potamus in health care data security," *CACM*, Vol. 47, Num. 7, pp. 25-28, http://doi.acm.org/10.1145/1005817.1005840 , July, 2004.

[12] Microsoft, Inc., *Windows XP system restore*, http://msdn.microsoft.com/library/default.asp?URL=/library/techart/windowsxpsystemrestore.htm, April, 2001.

[13] Microsoft, http://msdn.microsoft.com/library/default.asp?url=/library/enus/vbcon/html/vbcon IntroductionToFileSystemComponents.asp, 2004.

[14] *Policy Enforcement tools, McAfee System Protection – McAfee Policy Orchestrator,* http://www.networkassociates.com/us/products/mcafee/mgmt_solutions/epo.htm .

[15] King, Samuel T. and Peter M. Chen, "Back Tracking Intrusions," *Proceedings of the 19th Symposium on Operating System Principles (SOSP 2003)*, Bolton Landing, NY, October, 2003.

[16] Sarbanes-Oxley Act of 2002, HR 3763, PL 107-204, 116 Stat 745, United States Code, 2002, codified in sections 11, 15, 18, 28, and 29 USC.

[17] *Sygate – Policy Enforcement,* http://www.sygate.com/solutions/policy-enforcement.htm .

[18] Du, Wenliang, Aditya P. Mathur, Praerit Garg, "Security Relevancy Analysis on the Registry of Windows NT 4.0," *Proceedings of the 15th Annual Computer Security Applications Conference*, December, 1999.

[19] *Windows NT Workstation Resource Kit – Windows NT Registry*, http://www.microsoft.com/resources/documentation/windowsnt/4/workstation/reskit/en-us/24_reged.mspx .

[20] Wang, Y., C. Verbowski, J. Dunagan, Y. Chen, H.J. Wang, C. Yuan, and Z. Zhang, "STRIDER: A black-box, state based approach to change and configuration management and support," *Proceedings of the USENIX LISA Conference*, October, 2003.

Herding Cats: Managing a Mobile UNIX Platform

Maarten Thibaut and Wout Mertens – Cisco Systems

ABSTRACT

Laptops running UNIX operating systems are gaining market share. This leads to more sysadmins being asked to support these systems.

This paper describes the major technical decisions reached to facilitate the pilot roll out of a mobile Mac OS X platform. We explain which choices we made and why. We also list the main problems we encountered and how they were tackled.

We show why in the environment of a customer support organization at Cisco, a file management tool with tripwire-like capabilities is needed. Radmind appears to be the only software base meeting these requirements. We show how the radmind suite can be extended and integrated to suit mobile clients in an enterprise environment.

We provide solutions to automate asset tracking and the maintenance of encrypted home directory disk images. We draw conclusions and list the lessons learnt.

Introduction

Mobile UNIX Platforms

People have been using UNIX for more than 25 years. For much of that time computers running UNIX were large, clunky machines that could only be called mobile if you happened to own a truck. The physical location of the computer and its network address remained the same for much of the machine's lifetime.

Today we are faced with mass-produced UNIX laptops that move around the world as fast as their users do.[1]

Solutions specific to UNIX machines need to be put in place to handle UNIX laptop deployments in the enterprise.

About Us

Cisco's [2] Technical Assistance Center (TAC) is a worldwide technical support organization. The TAC provides technical support to Cisco customers. We, the authors of this paper are members of the IT support group for the technical support engineers of the TAC.

All development, testing and debugging of networking products at Cisco is done on UNIX platforms – mostly Suns [5] running Solaris. Engineers need UNIX because of the large amount of utilities (some of them internal) available only for UNIX platforms.

In the last few years some of our users had been given Windows laptops, many of them had installed Cisco's version of Linux [3] on them in order to get access to the UNIX tools they needed. Recently our support group noticed a shift towards Apple [1] PowerBooks. It became clear that a supported mobile platform with UNIX capabilities was something our user base would greatly appreciate.

[1]Some laptops move faster than their users due to incorrect baggage handling or theft.

Additional Material

You can find additional material at http://radmind.org/contrib/LISA05. There you'll find:

- The radmind extensions discussed in this paper (distributed as patches)
- The scripts called by cron to update the system (radmind-update) and some scripts that toggle radmind-update features
- The login scripts enforcing the use of FileVault
- The asset database code (client and server)
- Various tidbits and scripts that could be of use to other system administrators
- MCX configuration examples

Choices

Constraints

The following constraints are imposed on us by the department using our services:

- Our environment needs to be highly available (even the desktops). We can meet this requirement by keeping spare desktops around, clustering our servers and distributing our services.
- Users need access to UNIX tools
- Users need a mobile platform so they can work from home or while visiting customer sites
- Customer data must be protected (encrypted) when carried off-site

Practises

All Solaris client computers in our environment are identical. Users do not get privileges to install or delete software outside their home directories. While this may seem unnecessarily restrictive it allows our team to scale: identical systems mean identical problems and identical solutions; this enables us to make sure the desktop is highly available.

We decided to apply the same principle for the pilot of our new platform. This allowed us to manage the software loadsets installed during the pilot. However, some software couldn't be installed on all systems so some kind of modular setup looks more feasible for a full rollout.

Mac OS X Laptops

In order to choose the right platform a list of requirements was made and a technical analysis was performed. The platform scoring the highest in the analysis was piloted. If no show stoppers were found a wider beta rollout would be initiated to find any remaining issues before migrating the entire organization.

The method used to analyze the various platform choices was a weighted-score analysis. Each item in the list of requirements is given a weight factor. A high weight factor means the requirement is important. Each platform is then given a score for that requirement, higher scores mean a better result.

Fixed desktops scored low because of the lack of mobility. Thin clients have higher mobility but aren't truly portable. Windows [4] scored low on manageability and UNIX compatibility. Linux laptops seemed unfeasible due to the lack of support from laptop vendors.[2] Mac OS X scored high for user installable apps, hotplugging sound devices and screens and for its overall user interface.

So the results clearly suggested an Apple PowerBook platform, mainly because of the fact that it is the only viable portable UNIX platform in the list; portability and UNIX capabilities had high weights attached in the analysis.

Radmind

No matter what client platform is chosen, we need to manage its disk image. In the case of Apple the package management system doesn't meet our needs because:

- Most Mac OS X software is distributed as an application inside a compressed disk image. To install the software the user simply drags the application onto the /Applications folder. There is no package information stored in this case.
- Apple's package management system has no uninstall capability (even if the software is distributed as a package).
- There is no tripwire-like functionality to warn us about changes to the filesystem.

Many open source tools exist to create and distribute packages. They lack a tripwire [13] functionality. Radmind [12] was the only toolset that either fulfilled the requirements or could be easily altered to fulfill the requirements.

Radmind's tripwire capabilities [10, 16] trap changes to the filesystem caused by package installs on the Mac OS X machine. This allows us to use any method to install a package and record the changes made to the system in a transcript.

FileVault

All customer data must be encrypted when carried off-site. There are two ways this can be tackled:

1. Encrypt the entire hard disk. This makes sure no hard disk data is accidentally revealed but it also means that once the system has booted, all data is available. Note that this causes an increase in load times as you're encrypting the entire operating system.
2. Encrypt user data only. Obviously, there is less overhead in this scenario. It can be argued that such a modular approach is inherently more secure: as soon as the user logs out their data becomes unavailable.

No vendor-supported out-of-the-box solution was found for Linux. Apple's FileVault feature uses an AES-128 encrypted autosizing disk image to store the user's homedir. While it wasn't designed with double digit gigabyte sizes in mind it scales reasonably well in our experience. The only challenges we had were:

- There is no out-of-the-box way to enforce its use. We ended up scripting the forced creation and maintenance at login time.
- Passwords mysteriously change. This is easily fixed, provided you've installed a "master password"[3] that can unlock any filevault on the computer.
- The disk image needs to be compacted every once in a while, which takes a *long* time and can only be done at logout.

All in all we are quite happy with this solution.

Asset Tracking

There are good reasons to track a system and its users:

- the computer needs to be returned to the lease company when the lease ends
- knowing who to ask for the system when you need it back
- if the machine was used in some illegal act it helps if you know where it was and who was using it
- your organization requires recording of all user logins

The at the end of this section shows the attributes we wanted to know about.

Somehow we'll need to get mechanisms in place that give us the information we need.

At the time we couldn't find any off the shelf solution to manage our systems. We implemented our own software, AssetInterface. AssetInterface is a network client that stores asset information in a central database. The information is stored locally until the

[2]Note that Tadpole Computer [6] now offers both Linux and Solaris laptops. Sun also sells Solaris laptops.

[3]This can be configured in the Security preference pane in Mac OS X's System Preferences.

database server becomes available – i.e., when the client is on the intranet.

The information in the asset database is also useful for support; it provides us with a history of DHCP addresses for the machine. We can login to a user's machine without having to ask for the IP address.

Attribute	Meaning
primary user	The person normally using this machine and responsible for returning it
login user	The person actually using this machine
location	Where is the machine?
system image release	What is the version of the OS? Are all updates installed?
serial number	The hardware serial number of the computer
ip address	Current IP addresses of the computer (wireless, ethernet, . . .)
mac address	Current MAC addresses of the computer (wireless, ethernet, . . .)

Directory Services

Mac OS X can be configured to use a wide range of directory services such as NIS [7], LDAP [15] and Active Directory [8]. Since we had an existing LDAP infrastructure for our Solaris systems we used it for our Powerbooks too.

One noteworthy point is that we duplicated the LDAP tree layout of Apple's Open Directory http://www.apple.com/server/macosx/features/opendirectory.html product and pushed it into our existing LDAP server as a separate subtree. The regular tree contains the identification and authentication data. The Open Directory subtree contains authorization data encoded as MCX records.

This setup allowed us to use Apple's Workgroup Manager application to configure the MCX features, discussed further. This requires some schema changes on your LDAP server.[4] You can bind to any unmodified third party LDAP server but this leaves you without the MCX feature set. You can also setup your own OpenDirectory server just for MCX management, leaving authentication to, e.g., Active Directory.

Managing the System State

There's more to managing a system than just controlling the contents of the harddisk. Strictly speaking, the system state consists of:

- the system hardware
- the state of all bits in memory and on the harddisk
- the CPU and hardware state

Luckily, we can control most aspects of the system though its harddisk image. For example,

- scripts run at boot time can change the PROM password or update firmware
- pre-apply scripts can unload kernel extensions before upgrading a kernel driver

File System Management

Radmind was conceived to help system administrators manage lab systems, it's typically run at logout or system boot. Scripts chain together the radmind tools to download loadsets, check for differences, apply changes, etc.

It works well on laptops because all network connections originate from the client.

In what follows we outline some of the changes we had to make to suit radmind to our needs.

Scripting

The clients run a script from cron to check for updates and install them. We divert from common practice in several ways:

- We run this script while the user is active – this allows updates to happen over VPN links. Contrary to popular belief, this approach caused very few problems.[5]
- Once updates are ready to install the user is notified of the pending update. They can then choose whether to allow the update to take place at that time. The notification to the user includes details such as the expected download size and whether the update requires a reboot. This allows the user to make an informed decision – which turned out to be very important:
  1. users need to feel empowered
  2. there may not be enough time for the update to install
  3. the user might be in the middle of something important and doesn't want to be disturbed
- Pre-apply and post-apply scripts are downloaded before any other loadsets and before the pre-apply scripts are run.
- When an update requiring a pre-apply script is pushed out, the pre-apply script can be bundled with the update. This is not the case with radmind which requires a three step scenario to accomplish the same task:
  1. push out the pre-apply script
  2. wait for all machines to receive the update
  3. push out the update that needs the pre-apply script

  Laptops can be out of touch for weeks on end so it's not practical to wait for step 2 to complete.
- The example script provided with radmind always runs a file system check. Ours only does

[4] LDAP administrators don't like making schema changes – make sure you ask nicely.

[5] Note that on Mac OS X, Software Update also installs software while the user is logged in.

so when there are new loadsets downloaded from the server – keeping clients' system load low.

- The process checking the file system is run at low priority to reduce user impact.
- We automatically create unique SSL certificates for each client machine. The server uses the common name (CN) inside the client's SSL certificate to decide which command file to present to the client. We can use this later on to setup test environments or to specify different system images for different platforms.[6] This also makes it easy to read and grep through syslog files (many SSL services log the CN of the client certificate). Automating the creation and installation of the unique SSL key is included in the asset tracking solution.
- Each application has its own transcript (and negative) so that:
  - a modular approach can be introduced later on (allowing the user to select which optional transcripts to install)
  - single applications can be easily added or removed without touching existing transcripts
- Config files for applications and the OS are kept in separate transcripts so that we don't lose the config files when we upgrade to a new release of the transcripted software.

Prebinding

Apple chose prebinding as a way to improve application launch times.[7]

On Mac OS X a typical binary uses functions from 10 or more different libraries or "frameworks," each of which typically binds to many more frameworks. The net effect is a slow system because of all the run-time dynamic binding that has to be done by the dynamic loader each time an application is launched.

One way to get around this is to do most of this work at installation time rather than at run time. The bindings are performed and the result is saved inside the binary. The next time the binary runs the bindings only have to be redone if the target of the bind has changed.

This clashes with a radmind-controlled installation because of the install-time prebinding: when you install an OS update that updates the C library, 80% of the executables change with it. When you upload these changes as a new transcript you end up overruling a large portion of the base loadset – though all that changed was the prebinding information inside those binaries.

We had to change Radmind so that it would calculate a file checksum that remains constant throughout prebinding changes. The patch is partly based on ctool[8] and is pending inclusion in the standard radmind distribution.

Apple will eventually move away from the current prebind method: in the future, prebind information will be stored in a separate directory maintained by the OS.

NetInfo

One thing you can't manage directly through radmind is NetInfo. It's a database that contains overriding values for all name services going from accounts to automatic mounts. It has the highest precedence and cannot be disabled. We ended up making a post-apply script that keeps the NetInfo database in sync with a template. This way we can use radmind to make changes to the system accounts and groups.

User Configuration Management

User Preferences

Most Mac OS X applications use the standard preferences storage system.[9]

This is a very clean system that allows the sysadmin to setup default values for any preference a program uses, using a series of overriding locations and a standard preference file layout. For example: an application called Bar developed by the Foo company stores its preferences in ~/Library/Preferences/com.foo.bar.plist. You can provide default settings by storing a file of the same name in /Network/Library/Preferences, /System/Library/Preferences or /Library/Preferences (the latter override the former).

If you want to override settings instead of just provide defaults you'll need to use the MCX infrastructure discussed below.

Unfortunately, not all applications use this preference system. These applications need to be managed in a more ad-hoc manner, if at all.

MCX

MCX (Managed Clients for OS X) allows control over many settings on OS X. For instance, you can specify which preference panes a user is allowed to open or override any preference for any application that uses Mac OS X's native preference framework.

- MCX also controls credential caching. You have to use MCX to make OS X store a local copy of a users' credentials, known as a Mobile User account.
- Portable Homedirectories also need to be setup with MCX.

You can find MCX configuration examples on our website. More information can be found at Macenterprise.org: http://macenterprise.org/content/view/61/42/.

---

[6]The radmind config file allows wildcards for the hostname comparison. We chose a common name that looks like "ppc7450.PowerBook15.W654398KQZ4.johndoe". For example, we could give a special image to *.johndoe or ppc7450.*.

[7]Several Linux distributions do something similar. In the Linux world, it's called "prelinking."

[8]Ctool calculates hashes of binaries. The checksum doesn't change when the prebinding on the binary is redone. See the website http://ctool.darwinports.com/.

[9]See Apple's developer website http://developer.apple.com/documentation/CoreFoundation/Conceptual/CFPreferences/ for details.

Supporting Infrastructure

Global radmind Deployment

For the server side of radmind, standard practice is to have one server that runs one radmind process on one port.

- In order to scale globally we have one radmind server per main site. They each serve files off a volume that is mirrored across all sites.
- The master volume is kept on a separate host that doesn't serve files. This makes sure we can make changes to the volume without interfering with the radmind server processes.
- We run three radmind instances per server: the stable, testing and staging releases. This allows us to test command files and radmind configurations before we change the stable release. Some of radmind's directories on the server can be shared between instances where appropriate.
- We created scripts for distributing changes from the master that make sure that all files are consistent before sending them to the slave servers.
- Changes to radmind's code can alter the layout and contents of the transcripts. In such cases we push the new radmind to the stable clients. In the same update we change the port number of the server where the new transcripts can be found. When all systems have upgraded in this hop-skip way, we upgrade the stable release and point the clients there again.

Backups

All relevant user data is regularly synchronized to the Solaris NFS server using a home grown rsync [14] front-end. The server data is periodically backed up to tape removing the need for a separate, native backup solution on Mac OS X. This also puts the data within easy reach on the NFS server where the users can get to it from their Solaris desktops. In our OS X 10.4 release we plan to use the built-in Portable Homedirectories instead of the rsync front-end.

Reporting

Some errors trigger an email report program (surprisingly named "mail-report") that automatically mails the admins with the error output. This email includes network configuration information allowing us to login to the system remotely and fix the problem before the user even notices it. We changed the mail configuration so it delivers the mail as soon as the user connects to the intranet. /etc/postfix/main.cf (postfix's main config file) is auto-generated from a template, this way we can fill in the hostname in the config file using a script.

Open Issues

Radmind Atomicity and Bandwith Use

Radmind's lapply phase downloads and installs files directly onto the live file system. It would be better if it downloaded all changes first and then applied them. This way updates don't fail somewhere in the middle when the user removes the network cable.

Radmind conserves bandwidth by only downloading changed files. The network connection to the server is not compressed, however. We are working on zlib compression for the data channel.

"Don't Care" Option

Radmind can provide defaults for a file. If the file isn't present on the system it is downloaded. If it *is* present its content is not checked – only its permissions are. We wanted a more versatile ignore command in radmind so some files would be ignored completely. A patch is available on our website.

Software Install

Some software needs to be installed on the root partition. Even if the user is given permissions to install applications there, the files will simply be removed at the next radmind-update. One possible solution is to record such installs using radmind. The loadset is added to those downloaded from the server, allowing local overrides to the base image. We hope to have this solution by the start of the next project phase.

Configuration Management

Full configuration management wasn't a priority during the pilot. We'll need to implement such a system later to cope with the many different setups and configurations that are possible. There are many projects that attempt to address issues like this, e.g., LCFGng [9] and CFengine [11]. LCFGng uses RPM packages to install and maintain the system. It can likely be modified to use radmind transcripts instead, adding a powerful security layer that can detect and fix issues automatically.

Lessons Learnt

- Most package management tools are not written with fault tolerance in mind. They assume that they know the state of the system without checking it. Even if they include a "package checks" tool, they don't allow you to restore missing or altered files.
- Tripwire-like tools operate on the filesystem – not on an assumed state of the filesystem. Because of this they are inherently fault tolerant. Other package tools should consider implementing a fast way to:
  - verify the filesystem contents versus the package descriptions
  - fix inconsistencies and compromised components
- OS X is a great platform if you don't want to give your users administrator privileges. Many tools and frameworks are in place (and stable) that give users control over their systems while

keeping the reigns firmly in sysadmin hands. On top of that, most third party applications can be installed without privileges by simply dragging and dropping them on the user account. In our case it was possible to give our users limited sudo capabilities only. This didn't seriously limit their ability to do their jobs.
- Radmind gave us the ability to see what horrible things some third party drivers did to our precious image and allowed us to compensate accordingly.
- Keeping transcripts clean and modular can be difficult but pays off when you upgrade or disable applications.
- Giving users the ability to select which software they want installed on their systems is important for a full roll out. However, a pilot project can afford not to bother with that and can provide the same system image to all clients instead: the needs of the many outweigh the needs of the the few – or the one.

Conclusion

A mobile Mac OS X platform can be adequately managed using radmind. Leased laptops require some form of asset tracking software – a solution had to be written in-house.

Author Information

Maarten Thibaut earned his license in electrical engineering from KIHO, Gent (Belgium) in 1995 and joined Cisco Systems in 1998, where he has been a lab and system administrator for Cisco's Technical Assistance Center. He has a variable amount of cats. Reach him electronically at mthibaut@cisco.com .

Wout Mertens earned a Master's degree in computer engineering from the Universiteit Gent (Belgium) in 2000 and upon graduation he joined Cisco Systems as a system administrator in the same group as Maarten. In his spare time he tinkers with computers, sings, beatboxes and cooks – usually all at once. You can reach him at wmertens@cisco.com .

Acknowledgements

Many thanks to Lee Damon and Bart Lauwers for reviewing and proofreading this paper.

Bibliography

[1] Apple computer, inc., http://www.apple.com/ .

[2] Cisco systems, inc., http:/www.cisco.com/ .

[3] Gnu/linux, http://www.linux.org/ .

[4] Microsoft Corporation, http://www.microsoft.com/ .

[5] Sun Microsystems, Inc., http://www.sun.com/ .

[6] Tadpole Computer, http://www.tadpolecomputer.com/ .

[7] *System and Network Administration 1990*, Sun Microsystems, Inc., March, 1990.

[8] *Windows Server 2003: Active Directory Infrastructure*, Microsoft Press, 2003.

[9] Anderson, P. and A. Scobie, *Lcfg – The Next Generation*, UKUUG Winter Conference, http://www.lcfg.org/doc/ukuug2002.pdf, 2002.

[10] Arnold, Edward R., *The Trouble with Tripwire*, Technical report, SecurityFocus, http://www.securityfocus.com/infocus/1398 , 2001.

[11] Burgess, M., and R. Ralston, "Distributed resource administration using cfengine," *Software: Practice and Experience*, Num. 27, 1997.

[12] Craig, Wesley D., and Patrick M. McNeal, "Radmind: The Integration of Filesystem Integrity Checking with Filesystem Management," *Proceedings of The 17th Annual Large Installation Systems Administration Conference (LISA 2003)*, San Diego, California, http://www.usenix.org/events/lisa03/tech/full_papers/craig/craig.pdf , October, 2003.

[13] Kim, G. and E. Spafford, "Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection," *Proceedings System Administration, Networking, and Security, III*, 1994.

[14] Tridgell, A. and P. Mackerras, *The rsync Algorithm*, June, 1996.

[15] Wahl, M., T. Howes, and S. Kille, *RFC 2251 – Lightweight Directory Access Protocol (V3)*, Technical report, The Internet Society, 1997.

[16] Wilson, G. Samuel, Jr., *Solaris 10 Filesystem Integrity Protection Using radmind*, http://www.sans.org/rr/whitepapers/detection/1617.php , Technical report, SANS Institute, 2005.

Open Network Administrator (ONA) – A Web-Based Network Management Tool

Bruce Campbell and Robyn Landers – University of Waterloo

ABSTRACT

This paper presents *Open Network Administrator* (ONA), a web-based network management tool. Network administrators interact with ONA over the web, and ONA interacts with routers, switches and access points, using telnet and/or SNMP.

ONA provides a common web interface to a number of different vendors' network products, and provides granular access control, permitting network administrators across multiple departments to manage a network in a consistent manner. In addition to typical day-to-day operations like changing switch port speed, duplex, VLAN, etc., ONA performs a number of other maintenance operations, such as automatically backing up switch configurations, maintaining traffic graphs, and sending device reboot/up/down alerts via e-mail.

ONA leverages some existing components, such as RRDtool [1] and CVSweb [2]. A summary of existing tools, both commercial and open source, is provided, with comparisons made to ONA.

At the University of Waterloo, where there is both a central IT group and many sizeable decentralized autonomous IT groups, ONA rapidly gained widespread enthusiastic acceptance.

Introduction

The *managed* network switch offers significant advantages, including the ability to control individual port settings, speed/duplex, VLAN, etc. But there is a caveat: managed switches need to be managed. For a small network, the web-based management tool included with most switches is adequate. But as the network expands, as IT staffing grows, and as the number of features packed into edge devices increases, a need to treat the network edge as a *system* develops. Managing each switch independently becomes unwieldy.

The movement of network features from the core to the edge has not been matched by a corresponding increase in availability of edge device management tools. The traditional approach to edge device management is to try to keep the edge simple, and do everything smart in the core. This leaves powerful features of modern switches unleveraged, and ultimately leads to scalability issues.

Abrahamson, et al. [15] point out the importance of integrating a definitive source of edge device configuration data into a software tool that is easy to use and of significant benefit to its users. Many network vendors do offer a comprehensive network management tool which can do just about everything imaginable... with that vendor's latest and greatest network equipment. The prospect of upgrading an enterprise's entire network infrastructure, and then being tied to a single vendor, often makes this option unrealistic.

ONA can be dropped into an organization's existing heterogeneous network, with existing equipment, and existing network management structure (or lack thereof), and provide an immediate benefit.

Motivation

In 2000, the networking for the Davis Centre building at the University of Waterloo was upgraded from thinnet ethernet to switched ethernet. The networking in this building was managed by the IT group for the Faculty of Mathematics.

Extreme Networks hardware was selected, and it was initially managed through the Extreme Networks management tool, EPICenter [3].

The building also hosted researchers and administrative staff from the Faculty of Engineering, which had its own IT staff. In order to streamline the day-to-day network management activities (i.e., moves, adds, and changes), a number of IT staff from both faculties involved were given administrative access to EPICenter. Procedures were developed for documentation and change notification.

EPICenter did not support the notion of shared management well, and logging was minimal. There was no way to restrict the types of changes each staff member could make, and tracing back changes to a specific person was difficult. The learning curve for EPICenter was fairly steep, and the user interface made it easy to make errors. This created a need to give EPICenter administrative access to as few IT staff as possible.

Conversely, with a major network upgrade in progress, the number of moves, adds, and changes was significant. This created a need to give EPICenter administrative access to as many IT staff as possible.

We needed a tool that permitted IT staff to make the changes they needed to make, and prevented them from making changes they shouldn't make.

Requirements

The minimal requirements were for a secure, easily usable and accessible tool to permit network support staff to change speed, duplex, VLAN, etc., settings on network switch ports for which they have been authorized, and to log all such changes.

The requirements for being secure and easily accessible made web-based operation the preferred choice. Although it is possible to install custom software on the desktops and laptops of all network support staff, or to use X11 or RDP to connect to a server-based GUI application, these are considered significantly less practical than web-based operation.

Prior to investigating a switch port management tool, we had already developed systems to perform a number of other network related tasks (e.g., maintaining traffic graphs with MRTG [4], sending device down/reboot alerts with Nagios [5]), as well as some custom scripts for saving switch configurations to a TFTP server. Each of these systems had its own list of switch names and SNMP community strings. This was not ideal, and led to the requirement that the new tool also perform these tasks, such that our existing systems could be phased out.

Finally, it was recognized that networking technology and the way we use and manage it is constantly evolving. This led to a requirement that the new tool be extensible to meet currently unforeseen needs. Therefore, while not strictly a requirement, an open source solution would be viewed as preferable to a commercial solution.

Alternatives

As of the day of writing, there is no web-based, open source, network management tool to be found. Most of the network management tools available are not in fact device management tools. The areas of DNS management, network monitoring, and traffic graphing appear to be well covered. They break out like this:

- LANdb, a network documentation tool, appears feature rich according to its web site, and even promises device management in a future release. However, development appears to have been stalled since 2000.
- GxSNMP is a device management tool, albeit not web-based. Again, development appears to have been stalled since 2001.
- Kiwi CatTools is a popular commercial Windows-based tool for managing, monitoring and reporting on network devices and their configuration. A limited version can be downloaded for free from their website, with more powerful versions aggressively priced.
- HP OpenView is perhaps the most widely known and powerful enterprise network management tool. Pricing is targetted towards the enterprise.
- Splat is a network device management tool with capabilities generally similar to ONA. The most obvious difference is that Splat is command-line oriented while ONA is web-based. ONA satisifies most of the requirements listed by the authors of Splat [15]. Splat uses RANCID [14] for secure communication with devices; ONA uses authenticated SNMP/telnet. Splat relies on an authoritative inventory database to obtain MAC addresses and hostnames. This is not practical for ONA because ONA is used by many departments in several faculties at UW, each of which might have its own inventory database design, or none at all. ONA implements features such as multi-level access control, automated switch recovery, and initial switch configuration, which were future goals for Splat.

Design

ONA is a web-based application, written in PHP [17]. It interacts with network switches, routers, and access points, using SNMP and/or telnet scripting. Device configuration and data are stored in a MySQL [18] database.

The database consists of approximately 40 tables that store definitions (administrators and devices), group memberships, state information (ARP and MAC tables), and logs. The **devices** table contains the device host names, device types (switch, router, wireless access point, etc.), primary group membership,

NAME	PURPOSE	INTERFACE	AVAILABILITY
cacti [6]	traffic graphing	web	open source
IP Manager [7]	DNS manager	web	open source
Kiwi CatTools [8]	switch manager	Windows GUI	commercial
LANdb [9]	network documentation	web	open source
WhatsUp gold [10]	network monitoring, mapping	web, GUI	commercial
HP OpenView [11]	network manager	web	commercial
GxSNMP [12]	network manager	Linux GUI	open source
OpenNMS [13]	network monitoring	web	open source
RANCID [14]	switch configuration	script	open source
splat [15]	switch configuration	script	open source
SNMPc [16]	network monitoring	web	commercial

Table 1: Existing management tools.

and SNMP community strings. The **administrators** table contains the userids, primary group membership, and preferences for each administrator. The sections on *Granular Access Control*, *Change Logging*, and *MAC/IP Querying, Logging, Searching* explain the group membership tables, state information tables, and logs, respectively.

ONA uses Apache [19] .htaccess for user authentication, and thus any underlying authentication mechanism (e.g., RADIUS [20], LDAP [21]) can be used. ONA receives the successfully authenticated userid from Apache, and does not interact directly with authentication servers itself.

While VLAN and port settings from all devices are stored in the ONA MySQL database, the configurations on the switches themselves are still considered authoritative. That is, if a switch configuration is changed manually, ONA will not overwrite that change, but rather will update its own database to reflect the change, either within 24 hours, or immediately if an ONA administrator chooses that option.

An alternative design which would make the ONA database authoritative was also considered. This could provide some advantages, such as the ability to make global configuration changes and have them automatically pushed out, or to replace a failed switch and have its configuration automatically updated. While the technical advantages were tempting, it would increase the risk that a bug in ONA could push a damaged configuration out to a large number of devices. Our experience with a commercial wireless access point manager has demonstrated this risk. We manually upgraded the firmware on a number of access points to fix a bug that was affecting our wireless deployment. The management software did not support the newer firmware, and rather than reverting to a "do not understand – do not do anything" state, it pushed an unusable configuration to the upgraded APs, putting them offline. This type of experience has the potential to chill enthusiasm, and was considered too risky for an environment that depends on voluntary adoption of solutions.

ONA is not a network documentation tool in itself. In addition to the standard port description field, ONA supports a comment field, which is stored in the ONA database. This field can be used to hold a jack number, hostname or other information as desired.

A word about terminology. An ONA *administrator* is someone who uses ONA to do network administration. An ONA *manager* is someone who has authority to manage or configure ONA itself (including for example granting permissions to administrators).

Main Features

Multi Vendor Support

ONA currently supports Extreme, HP Procurve, Nortel Baystack, Cisco 2900xl/3500xl and Cisco 2950/3550 switches, and Avaya AP-3 access points. Adding support for a new device involves writing PHP code to perform a number of primitive operations on the device, like fetching the port speeds and duplexes, setting tagged VLANs, etc. For example, the functions for managing the Nortel Baystack switch are as shown in Display 1.

The bulk of the work involved in adding support for additional devices is investigation of the SNMP involved. Beyond basic functions like querying the port statistics and interface descriptions, different vendors' SNMP implementations have little in common.

Each vendor typically relies on its own private MIBs, even for generic settings such as port speed and duplex. This is unfortunate as it reduces the opportunity to leverage existing code to support multiple vendors' equipment.

To give an idea of the effort required to add support for new devices, a student investigated the SNMP involved for the HP Procurve line over a period of two weeks, and then one of the authors developed the code, from scratch to debugged and in production, in about five days.

```
function set_nortel_port_tagged_vlans_via_snmp( $d, $portname, \
          $olduntaggedvlan, $untaggedvlan, $oldtaggedvlans, $taggedvlans )
function set_nortel_port_untagged_vlan_via_snmp( $d, $portname, \
          $oldvlan, $vlan, $istrunk )
function adjust_nortel_vlan_members( $d, $vlan, $remove_this_port, \
          $add_this_port )
function set_nortel_port_trunkmode_via_snmp( $d, $portname, \
          $trunkmode, $olduntaggedvlan, $untaggedvlan, $oldtaggedvlans, \
          $taggedvlans )
function get_nortel_vlan_configuration_via_snmp( $d, $signature )
function get_nortel_port_speeds_and_duplexes_via_snmp( $d, $signature )
function set_nortel_port_speed_duplex_via_snmp( $d, $portname, \
          $speed, $duplex )
function get_nortel_model_and_version_via_snmp( &$d )
function nortel_telnet_login( $d, $contin )
function nortel_telnet_logout()
function create_nortel_vlan_if_needed( $d, $vlan )
```

Display 1: Functions to manage the Nortel Baystack switch.

Traffic Graphs with RRDtool

RRDtool is integrated with ONA, and ONA automatically maintains RRD databases for all switch ports, via a cron job that runs every five minutes. Traffic graphs (or *traphs* as we like to call them) can be displayed in several formats. A summary traph page for each switch shows a 24-hour graph for each port. For each port, a number of traphs are available, ranging from a period of six hours to one year. A real time graph tool can show traffic on a specific port, updated every ten seconds. The real time traphs also show packet counts and error counts. An example summary traph page is shown in Figure 1.

Granular Access Control

Permissions are managed by means of relationships between administrators and devices. Administrators and devices are listed in tables that associate them with various groups. The groups are typically organizational units such as departments, faculties, IT support groups, and so on.

Devices and administrators each have a primary group to which they belong. The **devices** and **administrators** tables establish these primary group memberships. Additional group memberships are assigned through supplementary tables. Regular expressions may be used in these tables to specify, for example, ranges of ports, or devices whose names start with a common prefix, and so on. Each device (switch, router, or access point), switch port, VLAN tag, and administrator must be a member of at least one group.

Permission for an administrator to take a certain action on a certain device is determined by analyzing the group membership relationships. For example, an administrator may change the speed or duplex on a switch port if they have a group in common with the port or switch. They may change an untagged VLAN on a port if they have a group in common with the VLAN to which the port is being changed. If the port is a trunk port, then they also must have a group in common with all tagged and untagged VLANs on the port.

Every button in ONA and every settable parameter may be individually disabled or enabled on a per administrator basis. As an example of the level of granularity that this access control mechanism provides, an administrator could be given permission to change speed and duplex only, on one port only, and not have access to any of ONA's other features.

The following example illustrates how ONA uses groups and regular expression pattern matching.

- Two departments, "Engineering" and "Science," each have 10 switches.
- A core device "core-sw1" is managed by a department called "Central Services Department" (CSD)
- The Engineering switches are named eng-sw1 through eng-sw10
- The Science switches are named sci-sw1 through sci-sw10
- vlans 10 and 11 carry Engineering networks
- vlans 20 and 21 carry Science networks

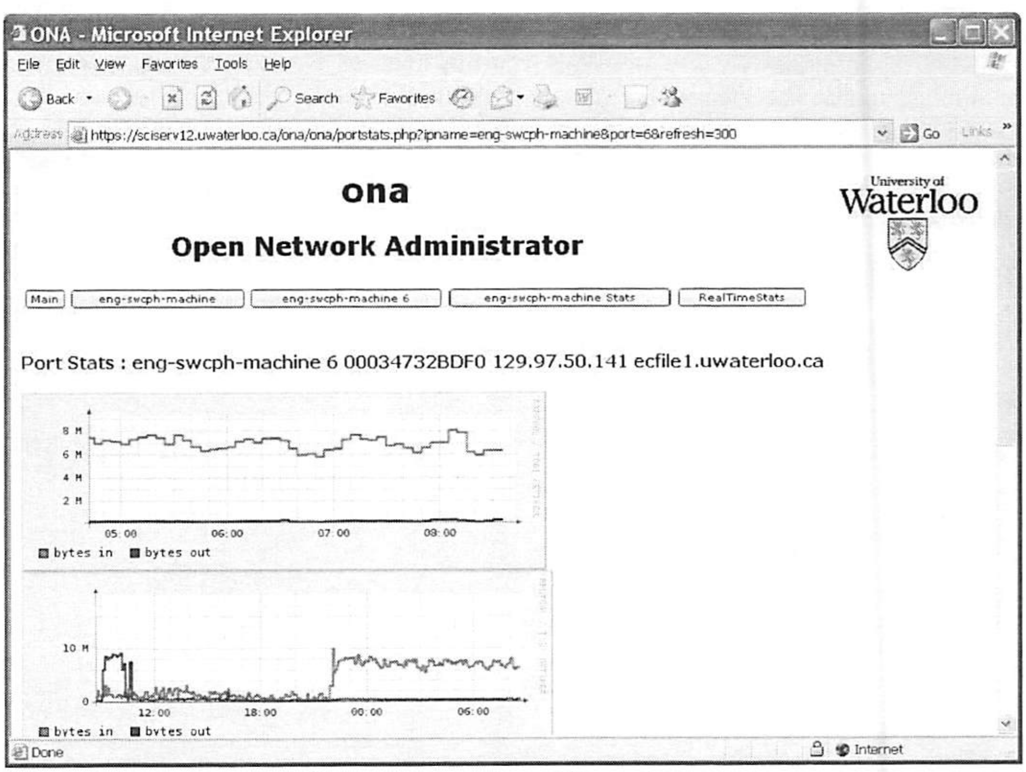


Figure 1: Summary traffic graph page.

- The userid "engadmin," who works in Engineering, has full access on all Engineering switches
- The userid "sciadmin," who works in Science, has full access on all Science switches
- A user "manager," who works in CSD, has full access on all switches
- A user "operator," who works in CSD, has permission to disable/enable client ports on all switches

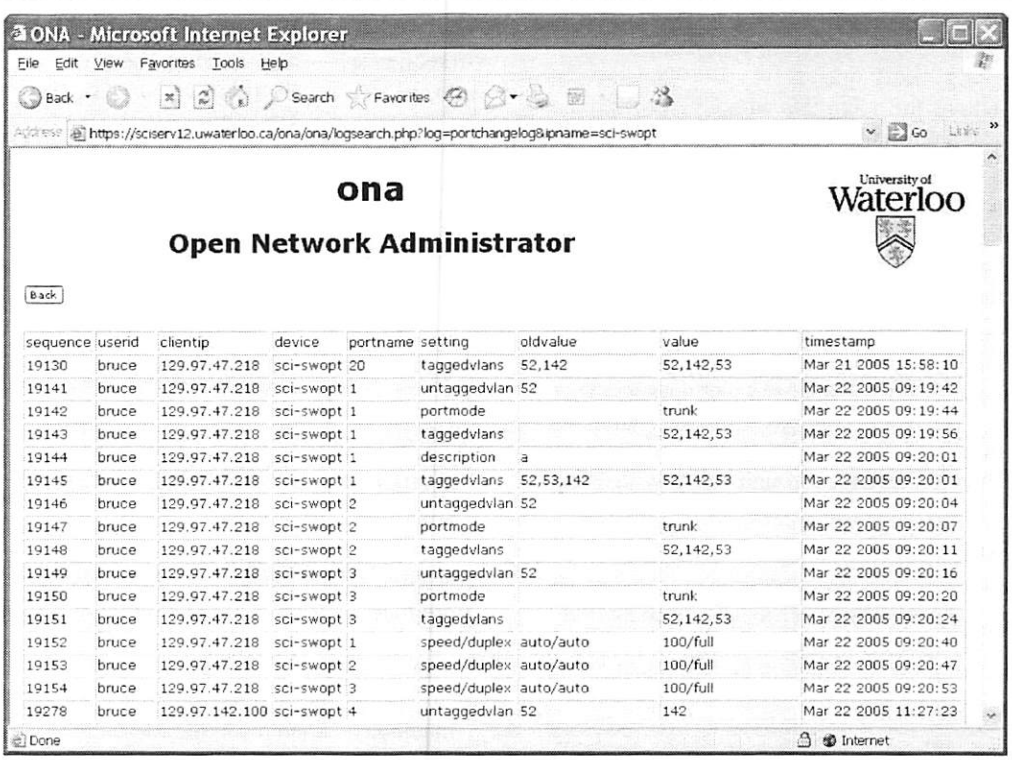
Implement the above using the managerial interface, as follows:

- Under "Devices," add eng-sw1 through eng-sw10, setting the groupid to "Engineering"
- Under "Devices," add sci-sw1 through sci-sw10, setting the groupid to "Science"
- Under "Devices," add core-sw1, setting the groupid to "CSD"
- Under "Administrators," add "engadmin," setting the groupid to "Engineering"
- Under "Administrators," add "sciadmin," setting the groupid to "Science"
- Under "Administrators," add "manager," setting the groupid to "CSD"
- Under "Administrators," add "operator," setting the groupid to "CSD," setting denytrunkchanges to "1," and setting allowededits to "portstate"
- Under "Device Group Memberships," add a groupid called "all," setting the ipname to "." (the regular expression consisting of a single period matches everything)
- Under "Vlan Group Memberships," add a groupid called "all," setting the vlan to "." (the regular expression consisting of a single period matches everything)
- Under "Administrator Group Memberships," add two entries, one for each of "manager" and "operator," setting the groupid to "all"
- Under "Vlan Group Memberships," add a groupid called "Engineering," setting the vlan to ^(10|11)$ (that is a regular expression which matches 10 or 11)
- Under "Vlan Group Memberships," add a groupid called "Science," setting the vlan to ^(20|21)$ (that is a regular expression which matches 20 or 21)

The above can be taken further. For example, engadmin and sciadmin could be given control over their respective ports on core-sw1, through use of "Port Group Memberships." Or, if Engineering and Science wanted to share switches in a given area, there are several ways to use group memberships to permit such sharing, without granting full access to non-shared switches.

Change Logging

For each change, the date, time, userid, IP address of the client, the parameter being changed, its previous value, and its new value, are all logged. For each device, a "Changes" button lists all changes in chronological order (See Figure 2). On the port edit screen, the changes for that particular port are displayed at the bottom of the screen. (All history is kept



sequence	userid	clientip	device	portname	setting	oldvalue	value	timestamp
19130	bruce	129.97.47.218	sci-swopt	20	taggedvlans	52,142	52,142,53	Mar 21 2005 15:58:10
19141	bruce	129.97.47.218	sci-swopt	1	untaggedvlan	52		Mar 22 2005 09:19:42
19142	bruce	129.97.47.218	sci-swopt	1	portmode		trunk	Mar 22 2005 09:19:44
19143	bruce	129.97.47.218	sci-swopt	1	taggedvlans		52,142,53	Mar 22 2005 09:19:56
19144	bruce	129.97.47.218	sci-swopt	1	description	a		Mar 22 2005 09:20:01
19145	bruce	129.97.47.218	sci-swopt	1	taggedvlans	52,53,142	52,142,53	Mar 22 2005 09:20:01
19146	bruce	129.97.47.218	sci-swopt	2	untaggedvlan	52		Mar 22 2005 09:20:04
19147	bruce	129.97.47.218	sci-swopt	2	portmode		trunk	Mar 22 2005 09:20:07
19148	bruce	129.97.47.218	sci-swopt	2	taggedvlans		52,142,53	Mar 22 2005 09:20:11
19149	bruce	129.97.47.218	sci-swopt	3	untaggedvlan	52		Mar 22 2005 09:20:16
19150	bruce	129.97.47.218	sci-swopt	3	portmode		trunk	Mar 22 2005 09:20:20
19151	bruce	129.97.47.218	sci-swopt	3	taggedvlans		52,142,53	Mar 22 2005 09:20:24
19152	bruce	129.97.47.218	sci-swopt	1	speed/duplex	auto/auto	100/full	Mar 22 2005 09:20:40
19153	bruce	129.97.47.218	sci-swopt	2	speed/duplex	auto/auto	100/full	Mar 22 2005 09:20:47
19154	bruce	129.97.47.218	sci-swopt	3	speed/duplex	auto/auto	100/full	Mar 22 2005 09:20:53
19278	bruce	129.97.142.100	sci-swopt	4	untaggedvlan	52	142	Mar 22 2005 11:27:23

Figure 2: History of device changes.

indefinitely. The need has not yet arisen to truncate history or roll older portions behind a "more" button.)

Managerial changes (e.g., adding a new switch, changing an SNMP community string), and batch telnet commands, are also logged, but these logs are not available for display through ONA at this time.

All ONA logs are stored in SQL tables. The structures of the adminlog, portchangelog and telnetlog are shown in Tables 2, 3, and 4.

These features make use of the above tables:

- The switch display screen shows MACs and IPs on each port. For trunk ports, the number of MACs seen in the last seven days, and a link to the complete list are shown (see Figure 3).
- The MAC/IP search tool shows all ports the MAC has been seen on (including trunks), with a quick link to the port if the MAC was found uniquely on that port. Also shown is the MAC, ARP, and port history. (e.g., if a network card was changed in a computer, ONA will show when that happened, both in terms of the change to the ARP table, and the MAC address present on the port); see Figures 4 and 5.

MAC/IP Querying, Logging, Searching

A cron job queries the MAC tables (and ARP tables for routers) regularly throughout the day. The data are stored in several tables, as shown in Table 5.

Switch Configurations and CVS Repository

A nightly cron job saves the switch configurations to a TFTP server, and can optionally push the configurations to additional TFTP/FTP servers. The configurations can also be placed in a tar ball, for automated pulling to network operations staffs' laptop computers. (Instructions for doing this with Cygwin [22] are included in the documentation for ONA.)

The pushing of switch configurations to alternate servers can be done based on the primary groups of the devices. For example, the configurations for switches in group "Engineering" could be pushed to a specific server belonging to Engineering.

Plain text device configurations are also saved to two different CVS repositories. One contains the raw configurations, and is not made available through ONA or CVSweb. The other contains the configurations minus sensitive information, such as community strings. The latter is available through ONA, with a link to CVSweb.

Managerial Interface

ONA managers (administrators with the "systemadmin" setting) can add/change/delete administrators and devices, and can adjust group memberships, etc., through the managerial interface, as shown in Figures 6 and 7.

Up/Down/Reboot Alerts

ONA logs and e-mails alerts when devices reboot, become inaccessible, or reappear. Administrators can disable receiving these e-mail alerts in the ONA preferences window. The set of devices about which a user receives notification is controlled by regular expression manipulation of the groups to which the devices belong.

FIELD	DESCRIPTION
sequence	sequence number
userid	userid of person who made the change
clientip	IP address of client used by userid
info	raw SQL (minus sensitive information) showing what was changed in the ONA admin tables
timestamp	date/time of change

Table 2: Fields of adminlog.

FIELD	DESCRIPTION
sequence	sequence number
userid	userid of person who made the change
clientip	IP address of client used by userid
device	name of device changed
portname	name of port changed
setting	settings, i.e., speed, vlan, etc.
oldvalue	old value
value	new value
timestamp	date/time of change

Table 3: Fields of portchangelog.

FIELD	DESCRIPTION
sequence	sequence number
userid	userid of person who made the change
clientip	IP address of client used by userid
device	name of device changed
announce	Boolean indicating whether this change should be announced in daily e-mail port (checkbox on telnet interface)
command	the telnet command that was run
timestamp	date/time of change

Table 4: Fields of telnetlog.

TABLE	DESCRIPTION
arp	IP to MAC
rarp	MAC to IP
arplog	changes in the ARP table
rawmac	MAC tables, with no analysis, i.e., raw data
mac	MAC tables, with results from trunk ports removed
maclog	changes in the MAC table

Table 5: Table names.

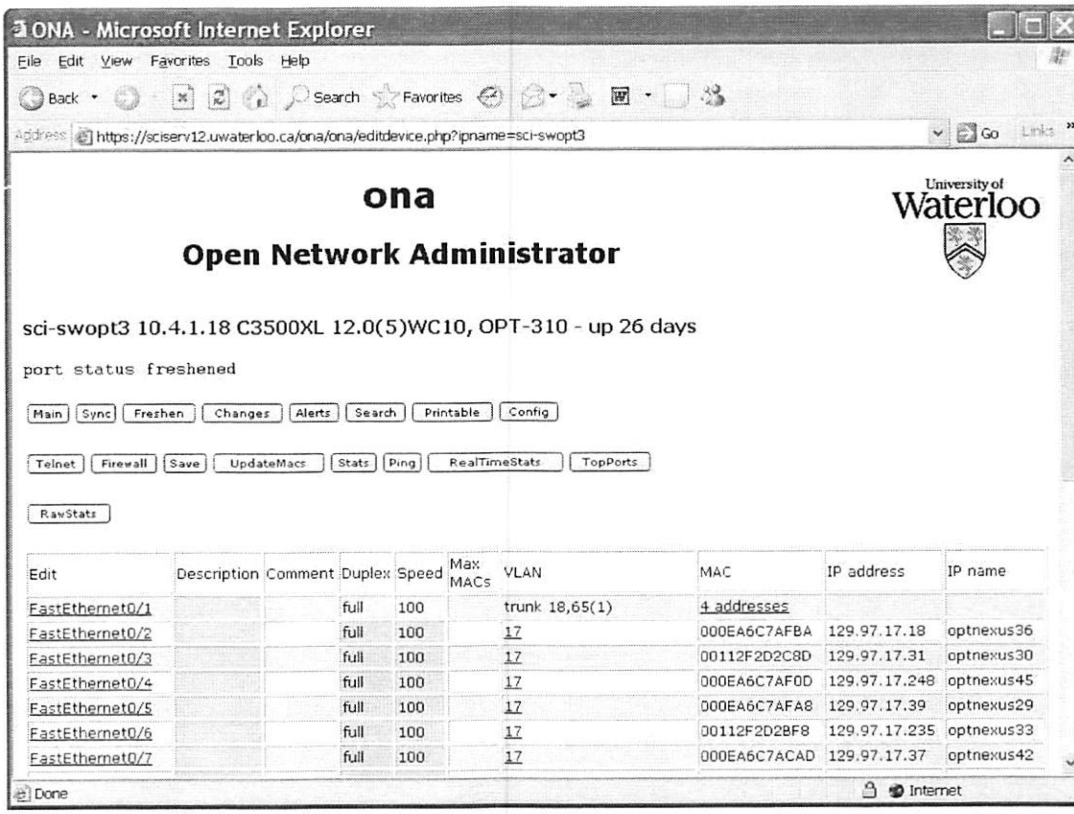


Figure 3: Switch page.

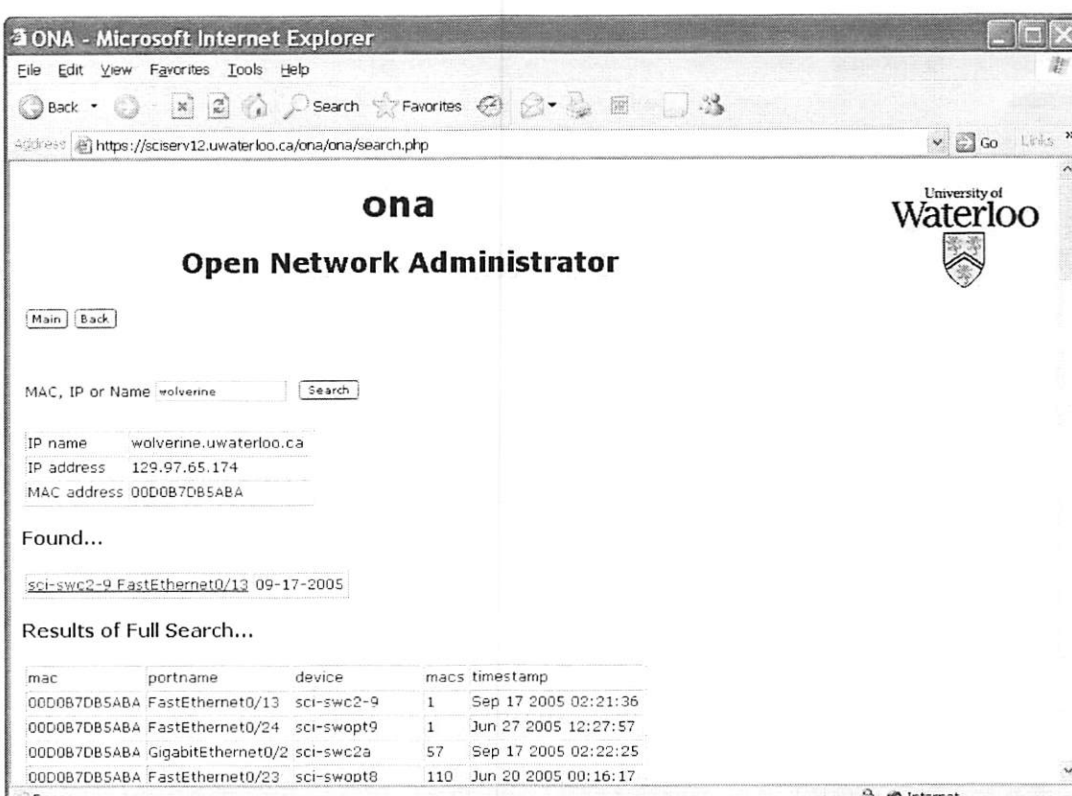
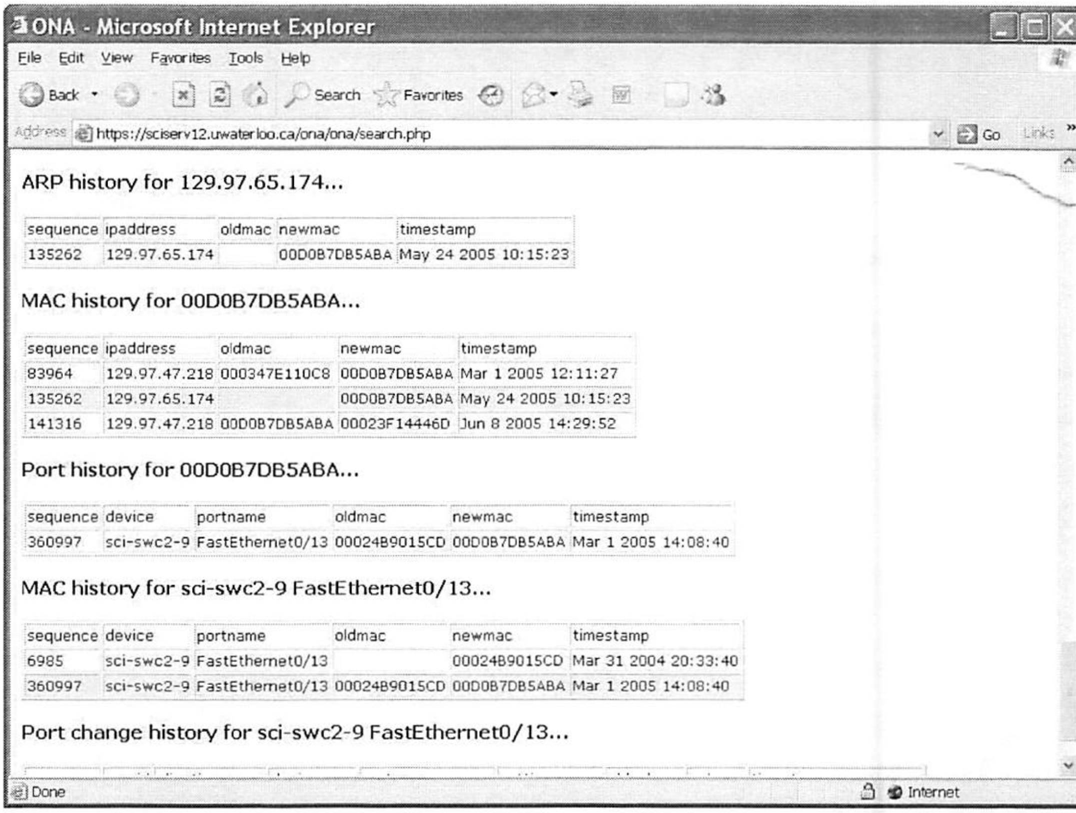


Figure 4: Search results.



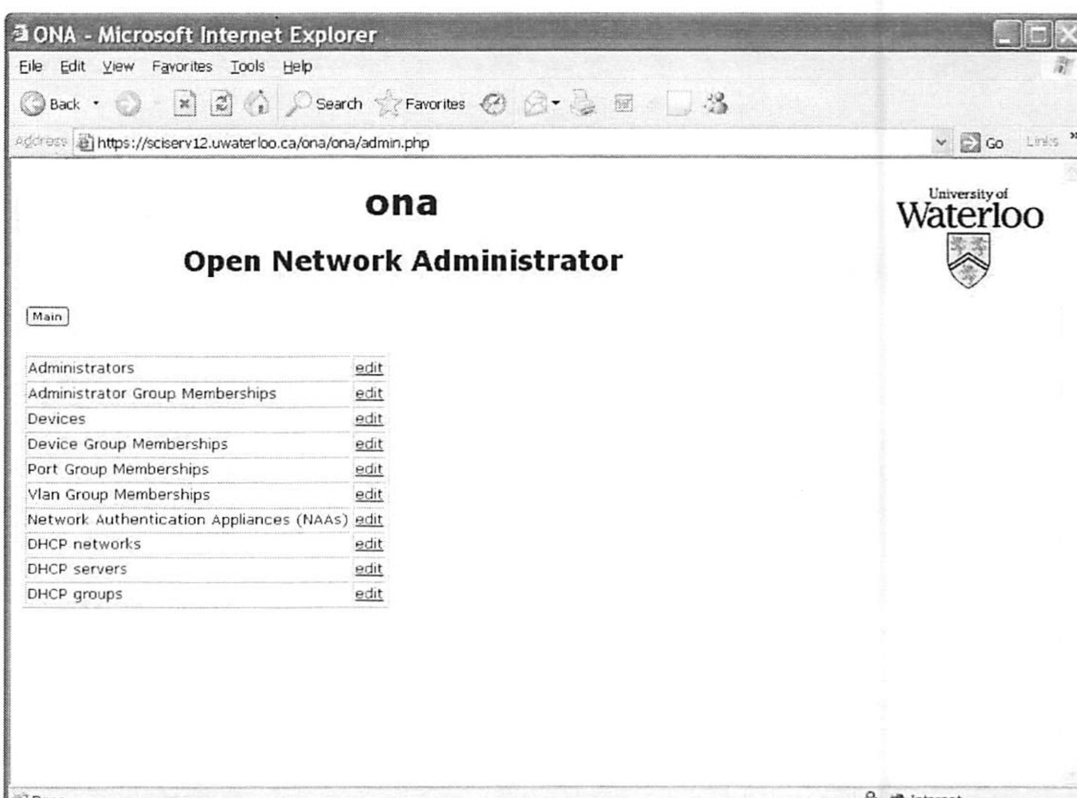
sequence	ipaddress	oldmac	newmac	timestamp
135262	129.97.65.174		00D0B7DB5ABA	May 24 2005 10:15:23

sequence	ipaddress	oldmac	newmac	timestamp
83964	129.97.47.218	000347E110C8	00D0B7DB5ABA	Mar 1 2005 12:11:27
135262	129.97.65.174		00D0B7DB5ABA	May 24 2005 10:15:23
141316	129.97.47.218	00D0B7DB5ABA	00023F14446D	Jun 8 2005 14:29:52

sequence	device	portname	oldmac	newmac	timestamp
360997	sci-swc2-9	FastEthernet0/13	00024B9015CD	00D0B7DB5ABA	Mar 1 2005 14:08:40

sequence	device	portname	oldmac	newmac	timestamp
6985	sci-swc2-9	FastEthernet0/13		00024B9015CD	Mar 31 2004 20:33:40
360997	sci-swc2-9	FastEthernet0/13	00024B9015CD	00D0B7DB5ABA	Mar 1 2005 14:08:40

Figure 5: History component of search results.



Administrators	edit
Administrator Group Memberships	edit
Devices	edit
Device Group Memberships	edit
Port Group Memberships	edit
Vlan Group Memberships	edit
Network Authentication Appliances (NAAs)	edit
DHCP networks	edit
DHCP servers	edit
DHCP groups	edit

Figure 6: Managerial interface – main screen.

Daily E-mail Report

Each night, ONA mails a summary of changes, alerts, and configuration differences from the previous day (minus sensitive information), to the administrators. Appendix 1 shows an example e-mail report.

Batch Telnet Interface

When specifically authorized in the **authgroups** table, administrators can perform batch telnet commands on a device or group of devices, for devices which support a command line interface. ONA is programmed to understand the telnet interfaces on the devices that it supports, so that it can supply the userid and password, transparently, and wait for command line prompts before sending the next in a series of telnet commands.

All commands run through the ONA batch telnet interface are logged. The administrator can choose whether to have telnet commands announced in the daily e-mail report, by use of a checkbox labelled Announce.

Commands run on a switch through telnet have the potential to make ONA's switch configuration database out of date. For example, if an administrator were to run commands via telnet to change the untagged VLAN on a port, ONA would continue to show the old VLAN for 24 hours, based on the values stored in its internal database. For this reason, the ONA telnet interface includes a radio button that can force ONA to synchronize its database immediately (Sync Now), or upon the next access to the switch through ONA's web interface (Sync Later).

Commands entered through the batch telnet interface bypass ONA's granular access control, so this interface is disabled by default, and is made available only to administrators who already know the actual switch passwords. An example of the use of this interface would be upgrading the firmware on a large number of devices, followed by rebooting the devices. The following commands were used to upgrade the firmware on approximately 100 Avaya access points:

```
download 129.97.50.121 \
               tmpdata/AVAP3.bin img
reboot 2
```

After entering the above commands into the command window, and selecting the 100 access points via checkboxes, ONA ran the command on all 100 access points with no further input required.

Configuration Translator

ONA can generate a suggested configuration fragment for a device using the text configuration commands of a different vendor. A manager may use this to relatively easily replace a switch with one from a different vendor.

The configuration fragment generated by ONA consists only of the VLAN database, and speed, duplex, description, and VLAN(s) of all ports.

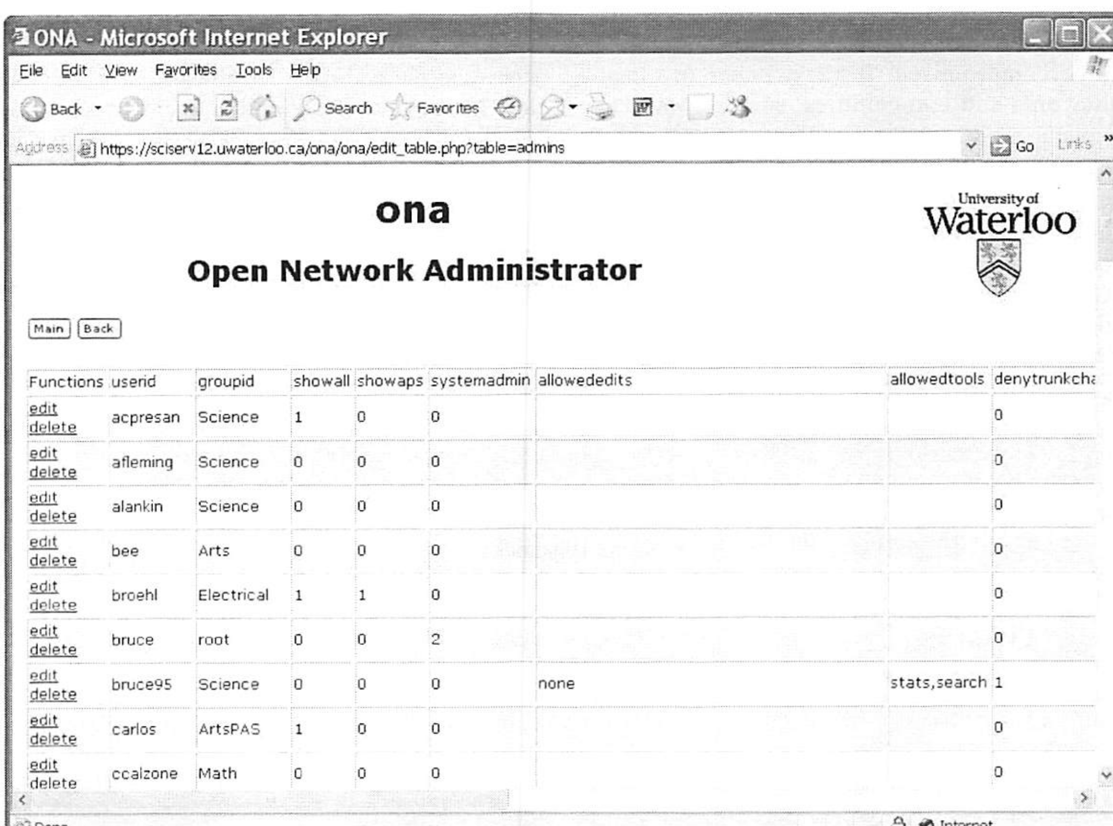


Figure 7: Managerial interface – editing administrators.

Display 2 shows sample configuration fragments for port 1, which is set 100/full, on VLAN 49.

The ONA Configuration Translator does not actually translate a given switch configuration text file from one vendor format to another. Rather, it generates the configuration fragment based on the VLAN and port information stored in the ONA database. For this reason, adding support for a new vendor involves writing fairly simple code to generate the text commands specific to that vendor, without knowledge of any other vendors' configuration file formats.

Security

Access to the ONA application is controlled through Apache .htaccess authentication. A configuration file contains the MySQL userid and password so that the ONA application may connect to the database.

The database contains the userids, passwords, and SNMP community strings for all devices. These are stored encrypted in the database, with a key contained in the configuration file.

ONA should be hosted on a server that does not permit general user login, or host web pages for general users, to help protect against unauthorized access to the configuration file and the database.

Access controls to limit administrative access should be applied on switches and access points, independently of whether one uses ONA. If a switch is stolen, the password or SNMP communities may be compromised. If all switches use the same passwords and communities, access controls restricting access to specific authorized hosts or subnets can reduce the impact.

Having a single network management tool handle a number of tasks means that userids, passwords and communities need be stored in only one location. This reduces the administrative burden of entering this information into several tools. By reducing this administrative burden, ONA makes it more likely that weak or widely known community strings will actually be changed, thus eliminating the common *I was meaning to get to that* explanation after a security compromise.

ONA's access controls limit what functions administrators can perform. Even in the case where administrators are given permission to do more than they actually need to do, ONA's logs indicate who did what and when.

ONA is also a handy tool when dealing with security incidents. When a computer is suspected of having a virus or is otherwise behaving badly on the network, ONA's "search" facility lets an administrator quickly search for the machine's connection point by either hostname, IP address, or MAC address. ONA shows the switch port on which the machine is connected, and one click takes the administrator to that port where it may be quickly disabled. A dialog box is available for the administrator to enter an explanatory message which is then e-mailed to the machine's user/owner/administrator as obtained by ONA from DNS.

Use of ONA At the University of Waterloo

Network management at UW is decentralized. The central IT group, Information Systems and Technology (IST), has responsibility for the external campus connection, internal campus backbone networking infrastructure out to points of presence in the faculties, and for all of student residence networking. The faculties may have responsibility for their own network infrastructure between their IST-provided point of presence and the end-user. Faculties such as Engineering and Mathematics have fairly large network infrastructures of their own; other faculties may have less or none (in which case IST provides it).

Before the development of ONA, network administrators used the vendor CLI tools for switch management. This meant learning the details of the tools for Cisco, Bay Networks (Nortel) and Extreme switches among others. Bay offered an ASCII GUI. Extreme offered both CLI and the EPICenter tool.

- Cisco:

```
interface FastEthernet0/1
speed 100
duplex full
switchport mode access
switchport access vlan 49
exit
```

- HP Procurve:

```
interface 1
   speed-duplex 100-full
exit
vlan 49
   name "cstclnet"
   untagged 1
exit
```

- Extreme:

```
configure port 1 auto off speed 100 duplex full
create vlan "cstclnet"
configure vlan "cstclnet" tag 49
configure vlan "cstclnet" add port 1 untagged
```

- Nortel Baystack:

```
interface FastEthernet1
speed 100
duplex full
exit
vlan ports 1 filter-tagged-frame enable
vlan ports 1 filter-untagged-frame disable
vlan ports 1 tagging disable
vlan create 49 name cstclnet type port
vlan members add 49 1
vlan ports 1 pvid 49
```

Display 2: Sample configuration fragments.

EPICenter was used for initial high level design but was abandoned due to its complexity and other shortcomings described earlier. Permission to modify switch settings was restricted to staff members who were directly responsible for network infrastructure.

ONA was well-received by senior network administrators. It offered a single easy interface to all the supported switches covering most of the device functionality. Significantly, it freed up senior network staff from the high volume of simple port configuration changes by enabling other technical staff to perform those simple changes themselves. Vendor CLI tools are still used in some circumstances when certain functionality is either not available via SNMP/telnet or not yet implemented in ONA.

ONA was developed in one department of the Engineering faculty, and was adopted voluntarily by most other Engineering departments, the Mathematics faculty, and others until at present four of the six faculties use it. In a decentralized environment where there are often tendencies towards preserving local control and resisting external influence, this widespread enthusiastic adoption of ONA is quite remarkable. The central IST department is now adopting ONA and plans are underway to bring all remaining campus network switches into ONA by the end of the year.

From the period May 1, 2004 to May 1, 2005, approximately 10,000 network port changes were made through ONA. ONA is the primary mechanism for switch port management, to the extent that the built-in web-based management tool that comes with the switches has been disabled, and is no longer depended on.

Even most veteran network operations staff, who are adept at the command line interfaces for multiple models of switches, choose to use ONA over telnetting into the switch directly.

Maintenance and Support for ONA

ONA was developed by one person, without the expectation in advance that it would be offered as an open-source tool to the community. It is implemented in roughly thirty PHP source files [23] and admittedly needs some polish to make it more readily amenable to both collaboration by multiple maintainers, and to outside use. However, it has been successfully installed by someone else without a great deal of difficulty.

ONA depends on having Apache web server, MySQL database, ModPHP (with SNMP, MySQL and FTP support enabled), CVSweb, and RRDTool installed. Then to bootstrap ONA, currently one must manually create the first manager and the default set of tables in SQL. (These bootstrapping steps are documented in detail [24].)

ONA is actively supported and developed by Bruce Campbell with contributions from Dawn Keenan of the Information Systems and Technology group at UW. With more recent perspective arising from ONA's enthusiastic acceptance at UW, and with the realization that there do not appear to be similar tools evolving elsewhere, there has emerged a small group of people spanning several UW IT groups who are interested in becoming active maintainers of ONA. Also, ONA is recognized at the administrative level of the University's IT organizations as an important tool worthy of a commitment of resources. We expect that these factors will lead to ensuring that ONA is maintainable and well-supported. Constructive feedback from readers of this paper would also help us move towards making ONA a well-packaged open-source tool.

System Requirements

ONA is written in PHP and has been tested on both FreeBSD and Linux. It should work on any UNIX platform.

To give some idea of the CPU etc. requirements to run an instance of ONA, some details of UW's production ONA installation are provided.

The system which runs http://ona.uwaterloo.ca/ is as follows:

- FreeBSD 4.11
- dual 2.80 GHz Xeon
- 512 MB memory
- 3ware SATA RAID

Devices:

- 277 switches (17288 ports)
- 299 wireless access points
- 6 routers

The cron jobs that query the port traffic statistics, and query the MAC address tables of all devices, place a moderate load on the server, with the load average typically sitting around 0.50.

The cron job that updates the RRD databases takes about seven minutes to query and update the port statistics for the approximately 17,000 ports.

The MySQL database is approximately 400 MB, the bulk of which is the rawmac table.

Conclusions

ONA seems to fill an important niche that is currently comparatively vacant, and seems to fill it well. It is a secure, easy-to-use, open source, web-based network switch management tool.

Its ready acceptance by both traditional command-line-oriented "power user" network administrators and other IT support staff, in a strongly decentralized setting, suggests that its advantages are very appealing. Its numerous features and the simplicity of its user interface lead staff to prefer it over traditional methods.

ONA is designed to be friendly to changes made to switches by means other than ONA. This benign co-existence permits deployment in a staged manner, in

which network operations staff can continue to manage switches directly as desired, and use ONA when they decide to. The deployment of ONA at UW did not require an overhaul of network management practices. ONA doesn't do everything – some operations, such as creating quality of service profiles, access control lists, etc., must still be done directly. But since ONA is comfortable with device changes made unbeknownst to it, ONA itself does not become an obstacle to using device features which it does not yet support.

ONA reduces the burden of network management by performing a variety of maintenance operations and handling the vast majority of typical switch port configuration changes.

It is surprising that there seem to be no other open source web-based edge device management tools available. We hope others will examine and use ONA, give us feedback as we move towards making it a well-formed open source tool, or perhaps be inspired to develop and contribute their own.

Availability

ONA is available for download at http://www.freebsd.uwaterloo.ca/twiki/bin/view/Freebsd/OnaInstallation [23]. Documentation is available at http://www.freebsd.uwaterloo.ca/twiki/bin/view/Freebsd/OpenNetworkAdministrator [24].

Author Information

Bruce Campbell is the Manager of the Science Computing department at the University of Waterloo. He has been at UW since 1984, initially as a systems administrator in the Engineering faculty. Bruce (the dirt-biker) is the developer of ONA, and can be reached at bruce@scimail.uwaterloo.ca or +1 519-888-4567 x6991.

Robyn Landers has been a UNIX systems administrator in the Math Faculty Computing Facility at the University of Waterloo since 1989. Robyn (the sport-biker) can be reached at rblanders@math.uwaterloo.ca or +1 519-888-4567 x2030.

References

[1] Oetiker, Tobi, *RRDtool*, Swiss Federal Institute of Technology, http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/.

[2] Fenner, B., H. Zeller, A. Musha, V. Skytta, *CVSweb*, http://www.freebsd.org/projects/cvsweb.html.

[3] *EPICenter*, http://www.extremenetworks.com/libraries/prodpdfs/products/epicenter.asp.

[4] Oetiker, Tobi, *MRTG*, Swiss Federal Institute of Technology, http://people.ee.ethz.ch/~oetiker/webtools/mrtg/.

[5] Galstad, Ethan, *Nagios*, http://www.nagios.org/.

[6] Berry, Ian, *cacti*, http://raxnet.net/.

[7] Sorrell, Chadwick, *IP Manager*, http://unix.freshmeat.net/projects/ipman/.

[8] *Kiwi CatTools*, http://www.kiwisyslog.com/cattools2.htm.

[9] Madden, John, *LANdb*, http://landb.sourceforge.net/about.shtml.

[10] *WhatsUp Gold*, http://www.ipswitch.com/Products/WhatsUp/index.asp.

[11] *HP OpenView*, http://www.openview.hp.com/.

[12] Estrella, G., J. Friedrich, L. Liimatainen, G. McLean, J. Schulien, C. Small, *GxSNMP*, http://www.gxsnmp.org/.

[13] *OpenNMS*, http://wiki.opennms.org/.

[14] Kilmer, H., J. Heasley, A. Partan, P. Whiting, A. Schutz, *RANCID*, http://www.shrubbery.net/rancid/.

[15] Abrahamson, C., D. Parter, M. Blodgett, A. Kunen, N. Mueller, "Splat: A Network Switch/Port Configuration Management Tool," *Seventeenth Large Installation Systems Administration Conference (LISA '03)*, p. 247, San Diego, CA, USENIX, http://www.usenix.org/publications/library/proceedings/lisa03/tech/full_papers/abrahamson/abrahamson_html/index.html, October 26-21, 2003.

[16] *SNMPc*, http://castlerock.com/.

[17] *PHP*, http://www.php.net/.

[18] *MySQL*, http://www.mysql.com/.

[19] *Apache*, http://www.apache.org/.

[20] *RADIUS – Remote Authentication Dial In User Service*, http://www.ietf.org/rfc/rfc2865.txt.

[21] *LDAP – Lightweight Directory Access Protocol*, http://www.ietf.org/rfc/rfc2251.txt.

[22] *Cygwin*, http://www.cygwin.com/.

[23] *ONA download*, http://www.freebsd.uwaterloo.ca/twiki/bin/view/Freebsd/OnaInstallation/.

[24] *ONA documentation*, http://www.freebsd.uwaterloo.ca/twiki/bin/view/Freebsd/OpenNetworkAdministrator/.

Appendix 1: Example E-Mail Report

```
Subject: ona - 1 admin changes, 2 reboots, 4 port changes, 10 config lines

1 admin changes
2 switches have rebooted
4 port changes
2 switches 10 config lines changed
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

1 admin changes...

#707  Jul 11 2005 09:52:21 vic      UPDATE devices
            SET 'ipname'=aco-swhh11 WHERE 'ipname' LIKE aco-swhh1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

2 switches have rebooted...

aco-swhh11                  last reboot was at Mon Jul 11 6:56:37
dccore-exsw05               last reboot was at Mon Jul 11 10:23:27
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

4 port changes...
 seq       date/time          userid     switch      port    setting        value
______ ___________________ ______ ____________ _____ ____________ ____________
#26378 Jul 11 2005 09:58:14 bee      aco-swpas6 Fe0/7 speed/duplex     100/full
#26453 Jul 11 2005 14:13:20 peregi dccore-exsw20 5:4   description     5:4 ecenet
#26454 Jul 11 2005 14:13:20 peregi dccore-exsw20 5:4       comment DC-3579 A34 29
#26455 Jul 11 2005 14:13:21 peregi dccore-exsw20 5:4  untaggedvlan             90
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

2 switches 10 config lines changed...

#  aco-swpas6 changes   +3  -1
  !
  ! Last configuration change at 18:56:25 EST Sat Jun 18 2005
- ! NVRAM config last updated at 15:52:54 EST Thu Jul 7 2005
+ ! NVRAM config last updated at 08:58:11 EST Mon Jul 11 2005
  !
  version 12.0
  no service pad

   switchport access vlan 64
  !
  interface FastEthernet0/7
+  duplex full
+  speed 100
   switchport access vlan 64
  !
  interface FastEthernet0/8
==========================================================

#
#  dccore-exsw20 changes   +3  -3
#

- # Alpine3808 Configuration generated Fri Jul 8 22:43:08 2005
+ # Alpine3808 Configuration generated Mon Jul 11 22:46:58 2005

- configure vlan "VLSINet" add port 5:4 untagged

+ configure vlan "ECENet" add port 5:4 untagged

- configure port 5:4 display-string "5:4 vlsinet"
+ configure port 5:4 display-string "5:4 ecenet"
```

An Open Source Solution for Testing NAT'd and Nested iptables Firewalls

Robert Marmorstein and Phil Kearns – The College of William and Mary

ABSTRACT

As firewalls have increased in power and flexibility, the complexity of configuring them correctly has grown significantly. An error in the firewall configuration can compromise the security of the system or interfere with normal network activity. The chance of an error increases when coordinating multiple firewalls, because the interaction between filters may hide errors more easily noticed on a single firewall. Firewalls on many networks use network address translation, which further increases the complexity of the firewall policy and creates additional opportunities for errors. Because errors in the firewall configuration are often extremely costly in time and security, system administrators need tools for verifying and debugging their firewall policy. ITVal is a tool for analyzing iptables-based firewalls that provides a plain English query language for simple firewall analysis. In this work, we describe extensions to ITVal that allow it to process network address translation rules and analyze multiple firewalls connected sequentially.

Introduction

Networks with a large number of hosts must defend against both external and internal intruders. While a perimeter firewall will block many external threats, it is useless against attacks from inside the network. With Trojan horses and viruses extremely prevalent, the problem of intrusions from internal hosts is growing rapidly [11]. To solve this problem, many system administrators complement the perimeter firewall with local firewalls on important internal hosts [10]. If the network is sufficiently large, the system administrator may also place additional firewalls between the perimeter firewall and groups of related hosts. The resulting architecture looks something like Figure 1, which depicts a network with a perimeter firewall, one unprotected host, two protected hosts, and a protected subnet. The protected hosts could be a mail server and a web server, while the protected subnet might consist of clients in an accounting department with financial information that must be secured. The perimeter firewall can mitigate denial of service and other external threats, while the firewall on each workstation secures services that must be protected from an inside intruder. Usually, most of the workstations have very similar filtering policies, which simplifies the distribution of changes to the policy, since the policy can be edited on a single administration host and then distributed across the network. One or more of the firewalls may also use network address translation(NAT) to further protect internal hosts or to work around the IPv4 address space problem.

The Linux firewall system, iptables, which provides NAT and stateful filtering, is well-suited for securing large networks of workstations and can be a

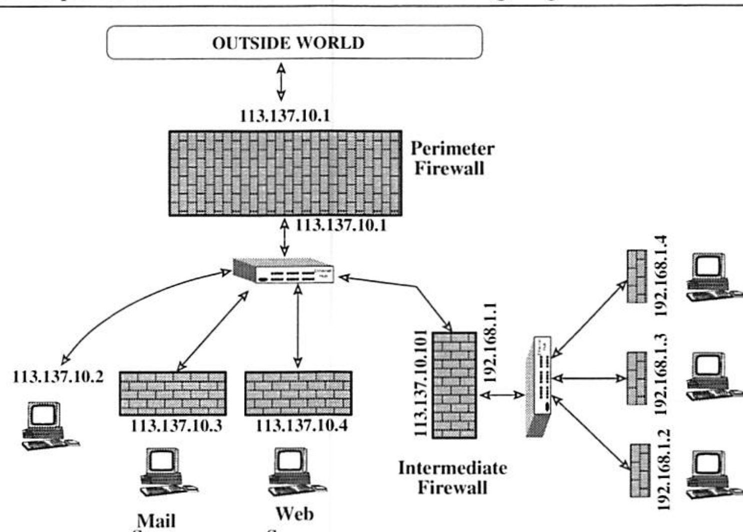


Figure 1: Common firewall architecture for defeating insider threats.

cheap solution for providing security against internal threats. Unfortunately, iptables rule sets are particularly difficult to understand and debug. Changes to the firewall may introduce subtle errors that disrupt normal traffic and can be difficult to diagnose.

The multiplicity of firewalls, in networks like the one described, greatly increases the difficulty of avoiding configuration errors. Removing a rule at the perimeter often means exposing hosts that are not sufficiently protected by their local firewalls. Incorrectly adding rules to a local or intermediate firewall can unintentionally block important network services. Firewalls with NAT are even more complex because a set of translation rules must be considered in addition to the filtering rules.

For example, consider the filtering chains given in Figure 2 that could be used to secure the network shown in Figure 1. The first chain is the forwarding chain of a perimeter firewall that protects a network 113.137.10.0/24 from intrusions on an insecure network 113.137.9.0/24. Rule 1 of the chain blocks traffic from the insecure network. Rule 2 protects the mail server by blocking all traffic from the outside world. The remaining rules secure various services that should be allowed to pass through the firewall. The second chain is the INPUT chain of the internal mail server, 113.137.10.3. It permits SMTP, secure IMAP, and SSH traffic, but blocks anything else.

There are several invariants that the administrator wishes to preserve about this network. First, web traffic from anywhere but the insecure network should always be allowed to host 113.137.10.4, the web server. Second, hosts from the outside world should never be able to SSH into the mail server. Third, no traffic should ever be permitted from the insecure network.

Let's assume that the administrator decides to allow SSH traffic through the perimeter firewall for hosts on subnet 113.137.8.0/24 by adding a rule to the

Chain FORWARD (Default Policy DROP)					
	target	prot	source	destination	flags
1	DROP	tcp	113.137.9.0/24	anywhere	
2	DROP	tcp	anywhere	113.137.10.3	
3	ACCEPT	tcp	anywhere	anywhere	TCP dpt:80
4	ACCEPT	tcp	anywhere	anywhere	TCP dpt:53

Rule Set on the Perimeter Firewall

Chain INPUT (Default Policy DROP)					
	target	prot	source	destination	flags
1	ACCEPT	tcp	anywhere	anywhere	TCP dpt:22
2	ACCEPT	tcp	anywhere	anywhere	TCP dpt:993
3	ACCEPT	tcp	anywhere	anywhere	TCP dpt:25

Rule Set on the Mail Server

Figure 2: Rule sets for an example network.

Chain FORWARD (Default Policy DROP)					
	target	prot	source	destination	flags
1	DROP	tcp	113.137.9.0/24	anywhere	
2	ACCEPT	tcp	113.137.8.0/24	anywhere	
3	DROP	tcp	anywhere	113.137.10.3	
4	ACCEPT	tcp	anywhere	anywhere	TCP dpt:80
5	ACCEPT	tcp	anywhere	anywhere	TCP dpt:53

Figure 3: An incorrect perimeter rule set.

Chain FORWARD (Default Policy DROP)					
	target	prot	source	destination	flags
1	DROP	tcp	113.137.9.0/24	anywhere	
2	DROP	tcp	anywhere	113.137.10.3	
3	ACCEPT	tcp	113.137.8.0/24	anywhere	
4	ACCEPT	tcp	anywhere	anywhere	TCP dpt:80
5	ACCEPT	tcp	anywhere	anywhere	TCP dpt:53

Figure 4: A correctly modified rule set.

forwarding chain of the perimeter firewall as shown in Figure 3. The first rule and the last three rules are the same as those in Figure 2, but the second rule is new. This change preserves the first and third invariant, but violates the second, because SSH traffic from the outside world can now reach the mail server. To correct the violation, she can either add restrictions to the filter on the mail server or switch the order of rules two and three in the perimeter filter.

A correct rule set for the perimeter firewall is shown in Figure 4. The new rule set allows HTTP and DNS traffic from 113.137.8.0/24, but preserves all three invariants.

Existing Tools

There is a fairly large body of tools available for testing firewalls. Port scanners, such as nmap [6], can be used to reveal open and closed ports on each host. Tools that perform general security audits, such as SATAN [5], Nessus [2], SARA [17], and ISS [9] also include components for testing firewalls by sending specially crafted packets to a host. In addition, there are a few tools, such as Ftester [3], designed specifically for analyzing stateful firewalls.

All of these tools are *active tools* that test the firewall by sending traffic through it. This has the disadvantage of consuming bandwidth and interfering with normal traffic. Furthermore, active tools are usually very inflexible. Rather than providing general functionality for investigating the firewall configuration, they are usually designed to test specific vulnerabilities. Also, they usually only simulate packets originating from a single host or small group of hosts. Incorrectly configured firewalls that allow packets from an untested host will pass the test even though an error exists. Some tools use address spoofing to mitigate this problem, but because of bandwidth constraints, no active tool can test every possible address that might originate a packet to the firewall.

More importantly, active tools do not work well with multiple firewalls and network address translation. Because packets dropped by one firewall are never seen by the second, it is often difficult for an active tool to generate a spoofed packet that will exploit configuration errors in both firewalls. Also, replies to packets with NAT'd source addresses may never been seen by the active analysis tool.

Because active tools have these drawbacks, *passive tools*, which perform an offline analysis of the firewall can be more practical. One such tool is the Lumeta firewall analyzer [18], a commercial product. Lumeta is based on Fang [1] and provides general query capability for Checkpoint and PIX firewalls. Because Lumeta provides an offline analysis of the firewall policy, it has many advantages over active tools. Unfortunately, Lumeta is not designed to work with iptables firewalls.

A few other groups have also done some work on passive analysis. A team at the University of Texas has developed a tool that uses SQL-like queries for firewall analysis [12]. They have also developed a system for improving the structure of the rule set using decision diagrams [7]. Another group has used decision diagrams to implement basic firewall queries [8, 16]. None of these tools are specifically targeted at iptables and they do not support NAT.

In previous work, we presented an open source tool, ITVal, for performing a passive analysis of a single iptables firewall. ITVal uses an efficient decision diagram library [13] to provide a plain-English query language that a system administrator can use to quickly test for vulnerabilities. ITVal is particularly useful for evaluating changes to the firewall configuration. A system administrator can perform an ITVal audit before and after each change to the firewall and examine the results to quickly determine if any of the important security invariants of the network have changed. As presented in [15], ITVal supported neither NAT nor analysis of multiple connected firewalls. In this work, we present extensions to ITVal that allow it to take these into account.

ITVal

ITVal implements a query engine for evaluating the configuration of a firewall. Some example queries are shown in Figure 5. The main components of each query are the keyword QUERY followed by an optional input chain, a subject, and a query condition. The input chain parameter determines whether the query should consider traffic inbound to the firewall, traffic outbound from the firewall, or traffic forwarded through the firewall.

The subject tells ITVal what information to report. For instance, the subject SADDY instructs ITVal to list the source addresses of packets that match the query. The condition is made up of primitives that can be combined with the logical operators AND, OR, and NOT to form complex queries. The available primitives are FROM, TO, ON, FOR, WITH, and IN, which specify source address, destination address, source port, destination port, TCP flag status, and connection state, respectively. There is also a special primitive, LOGGED, which matches packets for which the firewall contains a matching LOG rule. The reader is referred to [15] for the details of the query language. One feature of the query language we will exploit in our examples is the ability to name groups of hosts and services for use in multiple queries with the GROUP and SERVICE keywords, respectively.

The internal representation of both the queries and the rule sets is handled by the FDDL [13] decision diagram library. FDDL provides a data structure called a Multi-way Decision Diagram(MDD) for representing large sets of vectors compactly. MDDs are particularly well suited for representing firewall rules. In fact,

an MDD implementation of the iptables filtering algorithm showed significant performance gains over the existing implementation [4].

- QUERY SADDY TO 192.168.*;
  List all hosts with access to subnet 192.168.0.0/16.
- QUERY DPORT FROM 113.137.10.* AND NOT FOR TCP 993;
  List all destination ports, except the secure IMAP port(993), that can be accessed by hosts in the 113.137.10.0/24 subnet.
- QUERY SPORT NOT FROM 192.168.1.101 AND FOR 137.113.6.2;
  List all source ports open on host 137.113.6.2 to machines other than host 192.168.1.101.
- QUERY DADDY FOR TCP 25 AND (IN NEW OR IN ESTABLISHED);
  List all hosts that can receive packets on port 25 on a connection in the NEW or ESTABLISHED state.
- QUERY DADDY FROM 192.168.1.* AND (FOR TCP 25 OR FOR TCP 80 OR FOR TCP 110);
  List all hosts that can receive SSH, SMTP, or HTTP traffic from hosts on the 192.168.1.0/24 subnet.

Figure 5: Some example ITVal queries.

Formally, an MDD is a directed acyclic graph in which the nodes are organized into levels and every arc from a node at a level *k>0* points to a node at level *k-1*. In ITVal, each level of the MDD corresponds to one attribute of a packet potentially seen by the firewall. Every node of the diagram represents a set of packets that share some common attributes. Each arc at level *k* represents a choice of value for attribute *k*. An example MDD for the last query of Figure 5 is shown in Figure 6. Because we do not allow duplicate nodes with all arcs pointing to the same descendants, this means that a path through the MDD represents exactly one packet.

We use MDDs to represent both the rule set of the firewall and a set of queries. To represent a rule set, we add a level of terminal nodes to the bottom of the MDD which correspond to the ultimate fate of the packet (accepted or dropped) as determined by the rule set. To represent the queries, we instead add a level of terminals that indicate whether the packet matches the query conditions or not.

Intuitively, a rule set MDD represents the set of packets accepted by the firewall, while a query MDD represents the set of packets that satisfy the query conditions. When depicting MDDs graphically, we will often show only paths to the ACCEPT node or the MATCHES node. To save space, we also omit the terminal level.

To perform a query, we first represent the query and the rule set as MDDs. The MDD for the rule set is constructed from a rule set description generated using the "iptables -L -n" command. The MDD for the query is generated from a query file provided by the user. We then evaluate the query on the rule set by applying an MDD intersection operator to the two MDDs. The intersection of the two MDDs is the set of packets accepted by the firewall which match the conditions of the query.

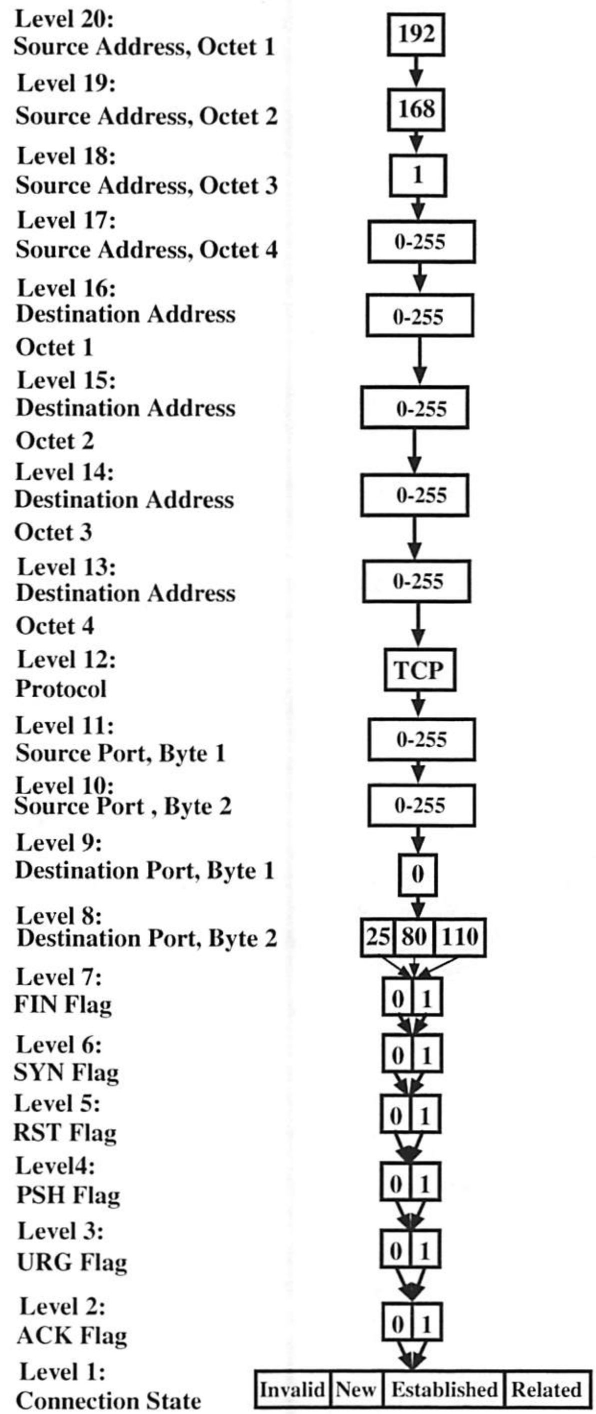


Figure 6: An example MDD.

In the following sections, we present the MDD-based infrastructure of the changes made to ITVal to

provide support for nested firewalls and NAT. The system administrator should have access to such information in order to increase his understanding of, and trust in, the tool.

Composing Nested Firewalls

In order to extend ITVal to work with multiple firewalls, we introduce the concept of a *meta-firewall*. A meta-firewall is an imaginary firewall that represents a composition of the rule sets of two or more serially connected firewalls. To analyze a meta-firewall, the user passes the names of the rule set description files on the command line. The order of the filenames must reflect the topology of the firewalls, with the innermost filter first on the command line and the outermost filter last.

The meta-firewall has three filter chains analogous to the three built-in chains of a normal firewall. The FORWARD chain of the meta-firewall regulates traffic passing through all the firewalls in either direction. The INPUT chain of the meta-firewall regulates traffic inbound to the innermost firewall through all of the outer firewalls. The OUTPUT chain represents traffic generated by the innermost firewall that successfully passes through the outer firewalls to the outside world.

Queries are performed against the meta-firewall as if it were a single iptables firewall. For instance, the query

```
QUERY FORWARD DPORT FOR 192.168.*
                        AND IN NEW;
```

will list the destination ports of packets bound for the 192.168.0.0/16 subnet that pass through all the firewalls in the set.

To construct the meta-firewall, ITVal joins the MDD for each chain of the component filters using the MDD intersection operator described in [15]. The algorithm is recursive and uses caching to improve performance. Using the caches, we are guaranteed to consider each pair of nodes in the MDDs only once. Because the performance depends only on the number of nodes in each MDD and not on the number of rules, this algorithm scales well to extremely large rule sets and queries.

Pseudocode for generating the meta-firewall is shown in Figure 7. The INPUT chain of the meta-firewall is constructed by intersecting the FORWARD chains of the outer *n-1* firewalls and the INPUT chain of the innermost firewall. The OUTPUT chain is created by intersecting the OUTPUT chain of the innermost firewall with the FORWARD chains of the outer *n-1* firewalls. The FORWARD chain is the intersection of all *n* FORWARD chains.

An MDD depicting the meta-firewall for the rule sets in Figure 2 is shown in Figure 8. In order to save space, only the levels for source address, destination address, and destination port are shown.

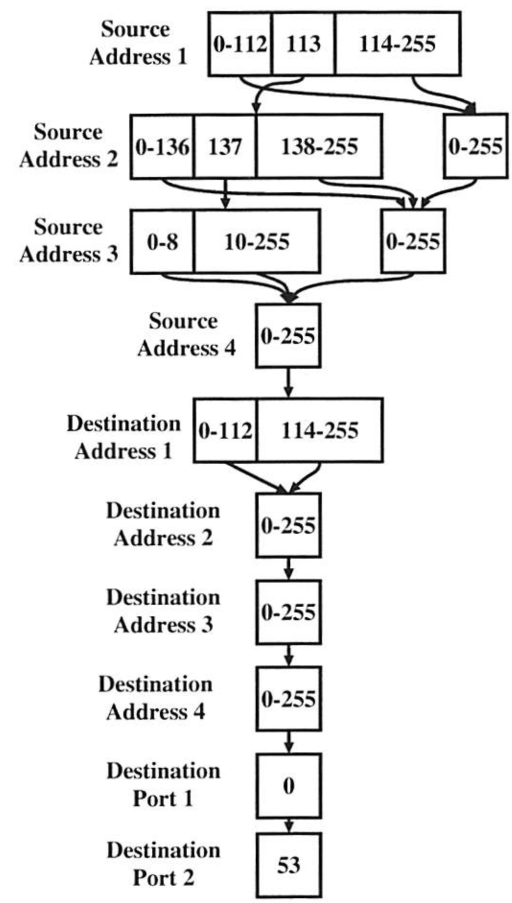


Figure 8: Combining two rule sets into a meta-firewall.

Figure 9 illustrates how ITVal might be used to detect the errors described in section 1. We depict the results of three queries before and after the incorrect

```
Firewall* ConstructFirewall(int n, Firewall* fws)
    newFW = NewFirewall()
    newFW.forward = fws[0].forward
    newFW.input = fws[0].input
    newFW.output = fws[0].output
    for i in 1 to n-1:
        newFW.forward = IntersectMDD(K, newFW.forward, fws[i].forward).
        newFW.input = IntersectMDD(K, newFW.input, fws[i].forward).
        newFW.output = IntersectMDD(K, newFW.output, fws[i].forward).
    return newFW.
```

Figure 7: Algorithm for constructing a meta-firewall.

change. The original, valid, results are shown in Roman font, while the later query results are shown in bold.

Each query corresponds to one invariant that the administrator wishes to preserve. The first query asks which hosts, other than those on the insecure net, can access the web server. In the original results, we see that, as expected, any other host can access the web server. In the modified results, we see that this important invariant still holds.

The second query asks whether the SSH port on the mail server can be accessed from outside the firewall. In the original results, no external machine can reach it. After the modification, however, the results show that the SSH port can be accessed from outside the firewall. SSH traffic from 113.137.8.0/24 can now reach the mail server. By comparing these results, the administrator will realize that she has made a mistake and take steps to correct it.

The last query tests whether services on the mail server are available to the insecure network. In both cases, the answer is no.

Network Address Translation

In addition to filtering packets that pass through the firewall, iptables provides a mechanism for modifying the destination or source address of a packet before and after filtering, respectively. This network

```
>ITVal Example.fw mail.rs mail.nat perimeter.rs perimeter.nat

#First invariant: Web traffic not from insecure net
#can always reach the web server
GROUP insecure 113.137.9.*;
QUERY SADDY INPUT FOR 113.137.10.3 AND NOT FROM insecure AND FOR TCP 80;

# Addresses: [0-112].*.*.*  [114-255].*.*.*  113.[0-136].*.*
# 113.[138-255].*.*  113.137.[0-8].*  113.137.[10-255].*
# 4294967040 results.

# Addresses: [0-112].*.*.*  [114-255].*.*.*  113.[0-136].*.*
# 113.[138-255].*.*  113.137.[0-8].*  113.137.[10-255].*
# 4294967040 results.

#Second invariant: External hosts should never be able to SSH to the mail
# server.
GROUP internal 113.137.10.*;
QUERY DPORT INPUT NOT FROM internal AND FOR TCP 22 AND TO 113.137.10.3;

# Ports:
# 0 results.

# Ports: 22
# 1 results.

#Third invariant: No traffic from the insecure network can reach
#the mail server.
QUERY DPORT INPUT FROM 113.137.9.*;

# Ports:
# 0 results.

# Ports:
# 0 results.
```

Figure 9: Query results before and **after** the change.

Chain PREROUTING (policy ACCEPT)					
	target	prot	source	destination	flags
1	DNAT	all	anywhere	113.137.10.101	TCP dpt:2002 to:192.168.1.2:22
2	DNAT	all	anywhere	113.137.10.101	TCP dpt:2003 to:192.168.1.3:22
3	DNAT	all	anywhere	113.137.10.101	TCP dpt:2004 to:192.168.1.4:22
4	DNAT	all	anywhere	113.137.10.101	TCP dpt:3000 to 192.168.1.2:9999

NAT rules for the intermediate firewall

Chain FORWARD (policy ACCEPT)					
	target	prot	source	destination	flags
1	DROP	tcp	113.137.10.4	192.168.1.0/24	

Filter rules for the intermediate firewall

Figure 10: Rule set of a NAT'ing firewall.

address translation (NAT) can also alter the destination and source ports of the packet. Properly handling NAT in the query engine is important, because the modified packet may be treated differently by the filtering rules than the original packet. In order for our queries to take NAT into account, we must modify the rule set MDD to reflect each of the NAT rules.

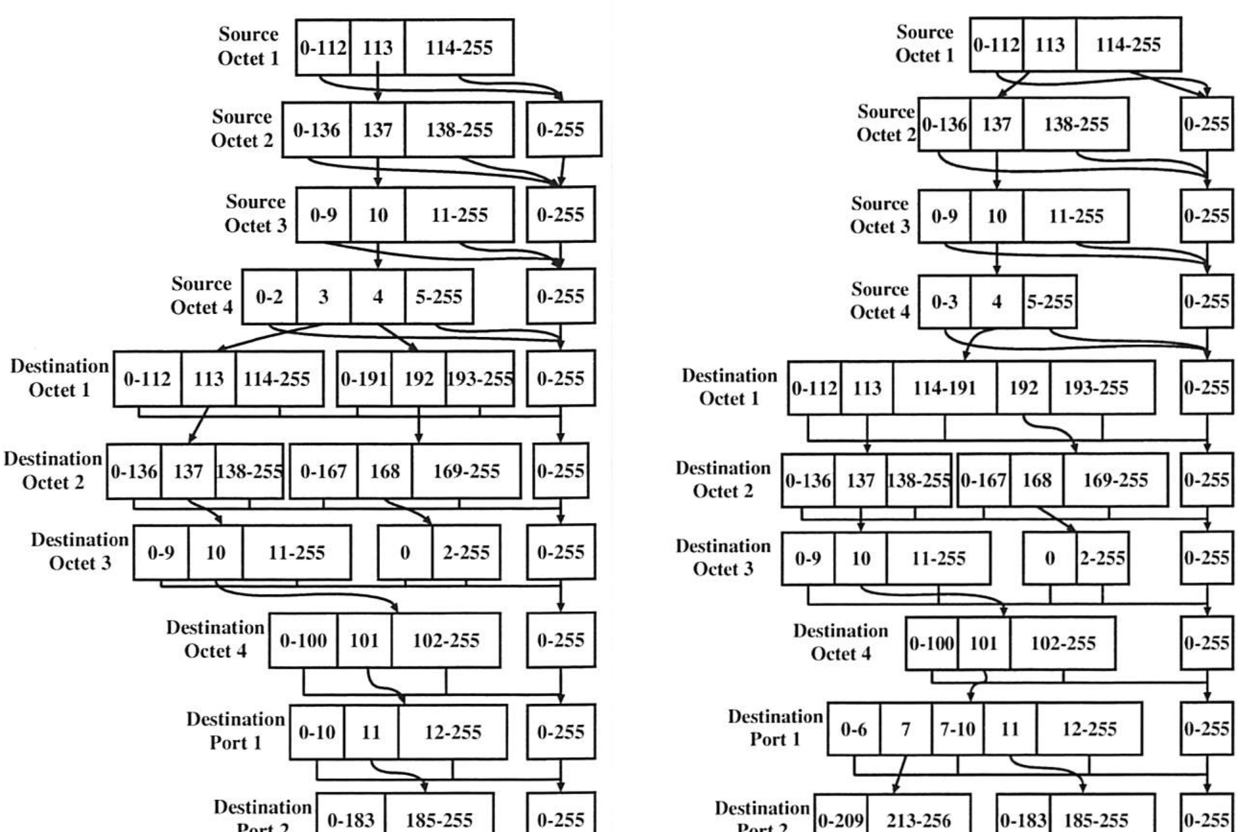
Like filtering rules, NAT rules are specified using chains. Destination NAT(DNAT) rules for incoming packets are specified in the PREROUTING chain, which is processed before filtering. In addition to the PREROUTING chain, iptables provides a POSTROUTING chain, processed after filtering, which is appropriate for source NAT(SNAT), and an OUTPUT chain for performing DNAT on locally generated packets.

An example rule set for a NAT'ing firewall, which might represent the intermediate firewall in Figure 1 is shown in Figure 10. This firewall protects an internal network 192.168.1.0/24 by hiding the addresses of hosts from the outside world. To access a host, an external system must connect to the NAT'ing firewall, which will forward the connection on the appropriate port. Each NAT rule has a domain and a range. The domain of the NAT rule specifies the set of packets that will be modified. In Figure 10, the domain of the first rule is "all TCP packets from 113.137.10.101 on port 2002". The range of the NAT rule specifies a set of new destination or source addresses and ports. In this case, the range of the rule is "for 192.168.1.2 on port 22". Because NAT can be used for primitive load balancing, the range may be a set of destinations rather than a single value. In this case, packets may be sent to any of the addresses in the range.

In the figure, the PREROUTING chain maps ports 2002-2004 to the SSH ports of internal hosts and also maps port 3000 to some proprietary database software running on port 9999 of machine 192.168.1.2. The FORWARD chain blocks traffic from the web server to those hosts.

Chain FORWARD (policy ACCEPT)					
	target	prot	source	destination	flags
1	DROP	tcp	113.137.10.4	192.168.1.0/24	
2	DROP	tcp	113.137.10.3	113.137.10.101	TCP dpt:3000

Figure 11: Incorrectly configured filter on the NAT'ing firewall.



Rule Set Before NAT Rule Set After NAT

Figure 12: Applying DNAT to the example rule set.

To access host 192.168.1.2 in the example a user should connect to port 2002 on 113.137.10.101. The firewall will then replace the destination address of the packet with "192.168.1.2" and the destination port with "22" before passing it through the filter rules and sending it to the router.

NAT adds another layer of complexity to the configuration of a firewall. One common mistake is to add filtering rules for the original address rather than the NAT'd address. For instance, if the system administrator decides to further restrict access to the internal hosts, she might add rule 2 of Figure 11 to prevent the mail server from accessing the proprietary database. While this blocks connections to the firewall itself, it does not block forwarded connections to the internal hosts. Unfortunately, such a mistake is very subtle and difficult to catch.

To model network address translation using MDDs, we create an operator which takes as inputs a NAT rule and the MDD representation of a set of packets. It produces as output an MDD representing the NAT-modified set of packets. Figure 12 shows the filtering rule set of the example before and after application of the DNAT algorithm. Tracing the MDD from the root node along the path for a packet from address 113.137.10.3 to port 3000 of host 113.137.10.101, we find that the packet will be accepted by the firewall. Pseudocode for performing DNAT on the MDD representation of the rule set of a single firewall is shown in Figure 13. SNAT is implemented similarly.

The algorithm takes as parameters a NAT rule and two MDD nodes at level *k*. The first node represents a set of unmodified packets. The second node represents the same packets after NAT has been applied. Initially, both parameters point to the root node of the rule set MDD. Our goal is to create a new set of rules which map each original packet to the target of its NAT'd equivalent.

Lines 1 and 2 check for the base case condition. If we have passed the levels which correspond to the destination port, we don't need to do any more work as all the NAT related information is contained in the preceding levels. We simply return the node representing the destination of the NAT'd packets.

Lines 3 and 4 check to see if the result has already been computed. If so, it is retrieved from the cache and returned.

Line 5 creates the result node. If the recursion has not yet reached the destination address levels, lines 6-11 set its arcs to the result of the recursive call. Otherwise, we consider each value *i* of the current attribute. If *i* is in the domain of the NAT rule, line 17 creates an arc to child of the NAT'd node. Otherwise, we point arc *i* at the child of node *p*.

Finally, in lines 20 and 21, we remove the node if it duplicates another node. If it is a duplicate, the index of the original node is added to the cache. Otherwise, the index of the result is added to the cache.

Figure 14 provides a query that might be used to detect the error in Figure 11. The group "insecure" is a list of hosts that should be prevented from accessing the secure server. A correctly configured firewall should always return an empty result. After making the incorrect change to the filtering rules, the system administrator adds 113.137.10.3 to the group. Now, if she again runs the query, she will see the results in Figure 15 and detect the error.

```
node_idx DNAT(NAT_RULE nr, level k, node p, node q)
1    if k<DPORTLEVEL:
2       return q.
3    if r = (k,p,q) is in the cache:
4       return r.
5    r = NewNode(k).
6        If k>DADDYLEVEL:
7             for each arc i <k, p>:
8                <k, r>[i] = DNAT(nr, k-1, pchild, pchild).
9             r = RemoveDuplicates(r).
10             Add (k, p, q) = r to the cache.
11             return r.
12    For each arc i ∈ <k, p>:
13        pchild = <k, p>[i]
14        if i ∈ Range(nr[k]):
15            for each value j in nr[k](i):
16                qchild = DNAT(nr, k-1, pchild, <k, p>[j]).
17                <k,r>[i] = Union(<k,r>[i],
                        DNAT(nr, k-1, pchild, qchild)).
18        otherwise:
19            <k,r>[i] = pchild.
20    r = RemoveDuplicates(r).
21    Add (k,p,q) = r to the cache.
22    return r.
```

Figure 13: Algorithm for destination NAT of MDDs.

```
GROUP insecure 113.137.10.3;
QUERY SADDY INPUT FROM insecure
AND FOR TCP 3000;

# Addresses:
# 0 results.
```

Figure 14: Query for detecting errors in the NAT'ing firewall.

```
GROUP insecure 113.137.10.3
113.137.10.4;

QUERY SADDY INPUT FROM insecure
AND FOR TCP 3000;

# Addresses: 113.137.10.4
# 1 result.
```

Figure 15: Results for an incorrectly configured firewall.

Nested Composition with Network Address Translation

In order to analyze a network that contains nested and NAT'd firewalls, the user specifies the filenames of filter rule sets and NAT rule sets alternately on the command line starting with the innermost firewall and working toward the outside.

The pseudocode in Figure 16 combines NAT with analysis of multiple firewalls. The procedure DNAT_ALL applies the chain of DNAT rules pointed to by its first parameter to the MDD specified by the second parameter. The procedure SNAT_ALL works similarly for SNAT.

In order to correctly derive the output chain of the meta-firewall, we work from the outermost firewall toward the innermost firewall combining pairs of firewalls. We DNAT the outermost firewall, then enter a loop in which we intersect the result with the unNAT'd filter rules of the next firewall to be considered. In each iteration, of the loop, we perform SNAT on the result of the intersection using the SNAT rules of the first firewall. We then DNAT using the DNAT rules of the second firewall. This alternating behavior simulates the traversal of a packet first through the PREROUTING chain, then through the filtering rules, and finally through the POSTROUTING chain.

To derive the input and forward chains, we perform the same operations in reverse order, working from the innermost firewall to the outermost firewall.

Conclusions and Future Work

Although modifying rule sets can be a complicated and error-prone process, the use of passive analysis tools can greatly simplify the task. Tools like ITVal that identify firewall configuration errors can prevent subtle mistakes from compromising the long-term security of a network. With our modifications, ITVal can be successfully used to analyze NAT'd firewalls in a hierarchical network topology. While this works well for networks in which the internal workstations share a common policy, developing algorithms for processing a more general topology is desirable.

In addition to supporting destination and source NAT, iptables provides two special case NAT targets. The REDIRECT target rewrites the destination address of a packet so that it will be routed to the firewall itself. The MASQUERADE target rewrites the source address of a packet so that it appears to have been originated by the firewall. The REDIRECT and MASQUERADE targets are extremely useful for environments in which addresses are assigned dynamically, since the address of the original host need not be known apriori when designing the rule set. In order to represent REDIRECT and MASQUERADE rules, ITVal needs to lookup the IP address of the host and perform SNAT or DNAT using the host IP as the new IP address. Although this is not currently implemented in the tool, we plan to add it soon. In order to perform MASQUERADE and REDIRECT, ITVal needs to know the IP addresses of each network interface.

```
Firewall* NAT(int n, Firewall** FW)
    newFW = NewFirewall()
    newFW.forward = DNAT_ALL(fws[0].dnat, fws[0].forward).
    newFW.input = DNAT_ALL(fws[0].dnat, fws[0].input).
    newFW.output = DNAT_ALL(fws[n-1].nat, fws[n-1].output).
    for i in 1 to n-1:
       newFW.forward = IntersectMDD(K, newFW.forward, fws[i].forward).
       newFW.forward = SNAT_ALL(fws[i-1].snat, newFw.forward).
       newFW.forward = DNAT_ALL(fws[i].dnat, newFW.forward).
       newFW.input = IntersectMDD(K, newFW.input, fws[i].forward).
       newFW.input = SNAT_ALL(fws[i-1].snat, newFw.input).
       newFW.input = DNAT_ALL(fws[i].dnat, newFW.input).
       newFW.output = IntersectMDD(K, newFW.output, fws[(n-i)-1].forward).
       newFW.output = SNAT_ALL(fws[(n-i)].snat, newFw.output).
       newFW.output = DNAT_ALL(fws[(n-i)-1].output, newFW.output).
    newFW.forward = SNAT_ALL(fws[n-1].snat, newFw.forward).
    newFW.input = SNAT_ALL(fws[n-1].snat, newFw.input).
    newFW.output = SNAT_ALL(fws[0].snat, newFw.output).
    return newFW.
```

Figure 16: NAT with multiple firewalls.

Since this information cannot be determined from the rule set, we will need to create an input mechanism for specifying these addresses.

Improving the output mechanism could also greatly improve the tool. Queries that return a large number of results can sometimes generate "information overload." Investigating ways to present this information more concisely, perhaps through a graphical tool, would greatly improve the utility of the tool. While system administrators can use ITVal to catch configuration errors, it is still sometimes difficult to identify the particular rules that cause the problem. Labeling arcs in the rule set and result MDDs could address this issue. Identifying the rules that cause a configuration error would also make it possible to perform guided repair of the firewall. Future versions of ITVal may not only catch firewall errors, but give a system administrator suggestions about how to change those errors to satisfy safety and liveness invariants of the network.

The latest version of ITVal is available on the web at http://itval.sourceforge.net/ .

Author Information

Robert Marmorstein is a graduate student at the College of William and Mary and a hard-core free software geek. When he is not hacking away at firewall analysis tools, he can usually be found tinkering with his assortment of Linux and BSD based systems. His e-mail address is rmmarm@wm.edu .

References

[1] Wool, Avishai, Alain Mayer, and Elisha Ziskind, "Fang: A firewall analysis engine," *Proceedings of the IEEE Symposium on Security and Privacy*, May, 2000.

[2] Anderson, Harry, *Introduction to Nessus*, October, 2003.

[3] Barisani, Andrea, "Testing firewalls and ids with ftester," *TISC Insight*, Vol. 5, 2001.

[4] Christiansen, Mikkel and Emmanuel Fleury, "An mtidd based firewall using decision diagrams for packet filtering," *Telecommunication Systems*, Vol. 27:2-4, pp. 297-319, 2004.

[5] Farmer, Dan and Wietse Venema, *SATAN: Security Administrator's Tool for Analyzing Networks*, 1995.

[6] Fyodor, "The art of port scanning," *Phrack*, 7(51), September, 1997.

[7] Gouda, Mohamed G. and Alex X. Liu, "Firewall design: Consistency, completeness, and compactness," *Proceedings of the International Conference on Distributed Computing Systems*. IEEE Computer Society, March 2004.

[8] Hazelhurst, Scott, "A proposal for dynamic access lists for tcp/ip packet filtering," *Technical Report TR-Wits-CS-2001-2*, University of Witwatersrand, April, 2001.

[9] "Internet Security Systems," *Internet Scanner User Guide, Version 7.0 SP 2*, 2005.

[10] Jonah, Kevin, "Multiple firewalls defend against multiplying threats," *Washington Technology*, Vol. 18, Num. 8, July, 2003.

[11] Leon, Mark, "Inside the Firewall: Will Bigger Encryption Keys Keep Your BI Data Safe From Harm?" *Intelligent Enterprise*, May, 2005.

[12] Liu, Alex X., Mohamed G. Gouda, Huibo Heidi Ma, and Anne HH. Ngu, "Firewall queries," *Proc. of the 8th International Conference on Principles of Distributed Systems (OPODIS-04), LNCS 3544*, Springer-Verlag, December, 2004.

[13] Marmorstein, Robert, *Designing and implementing a user library for manipulation of multiway decision diagrams*, MS Project Report, Department of Computer Science, The College of William and Mary, http://www.cs.wm.edu/Pubs/710paper.pdf, 2004.

[14] Marmorstein, Robert, *ITVal Website*, http://itval.sourceforge.net/ , 2005.

[15] Marmorstein, Robert, and Phil Kearns, "A tool for automated iptables firewall analysis," *FREENIX/Open Source Track, 2005 USENIX Annual Technical Conference*, pages 71-82, April, 2005.

[16] Fatti, Anton, Scott Hazelhurst and Andrew Henwood, "Binary decision diagram representation of firewall and router access lists," *Technical Report TR-Wits-CS-1998-3*, University of Witwatersrand, October, 1998.

[17] Todd, Bob, *SARA man page*, http://www-arc.com/sara/sara8.html .

[18] Wool, Avishai, "Architecting the Lumeta Firewall Analyzer," *Proceedings of the 10th USENIX Security Symposium*, August, 2001.

Towards Network Awareness

Evan Hughes and Anil Somayaji – Carleton University

ABSTRACT

Network and system administrators need to analyse network traffic for maintenance, security, and planning purposes. The volume of data on modern networks, however, make such analysis extremely difficult using existing open source tools. In this paper we argue that administrators need tools that will allow them to be more aware of the state of their networks, and we describe our vision for tools that would support such "network awareness" by analysing and visualising packet aggregations that are defined by both packet headers and payloads.

As a first step towards such tools, we have developed a library called `qcap`, a framework for packet and stream reconstruction that allows applications to tap packets at all layers of the network stack: from network, to transport, to the application layer. `qcap` is fast, able to process network data at speeds of 120 megabytes per second on commodity hardware; it is easy to use, providing a simple API that requires only a few lines of code to perform complex parsing tasks; and it is extensible, using BNF-like grammars to describe TCP protocols. We believe that `qcap` can provide the foundation for tools that will support greater network awareness for system administrators.

Introduction

The behaviour of computer networks is one of the great unknowns of computer science. Most network protocols are well known, the communicating hosts act (in some sense) on the behalf of their human masters, and network data are available at well known points in the network; nevertheless, we still cannot easily answer the question "what is the network doing?" High volume network traffic conspires with our lack of protocol reconstruction tools to obscure network content from us. Simple tasks require the construction of specialised software tools [15] to look for known or expected network events. Meanwhile, the flood of expected traffic obscures unexpected [8, 31] and possibly malicious traffic. The irony of this predicament should not be ignored: computers are information processors, and computer networks are designed to share information, and yet we do not have the means to understand the information processing that these human-made constructs are performing on our behalf.

General curiosity is not the only reason why we need to know what happens on computer networks as there are more pragmatic reasons to care. Network administrators need to fully understand the resources they control. Data flowing across the network dictates how the network should expand and be optimised; the more resources administrators can draw upon to understand the network, the more informed their decisions will be. Administrators also need to know the nature of network traffic for security reasons: when utilisation changes, they must be able to understand what has changed, and why. Further, other areas of computer science would benefit from a better understanding of network phenomena. For example, protocol designers must understand the environment their protocols are to inhabit. Additionally, network security researchers and practitioners need to understand the network and the effects of attacks.

The problem of analysing multiple high-bandwidth data streams in an online, complex environment exists in other contexts. For example, many researchers have recognised that complex displays and multiple alarm signals can reduce the "situational awareness" of pilots and other operators of complex equipment, making them more prone to errors [13]. Similarly, network administrators are distracted by voluminous logs and detailed packet dumps, all of which give important information, but none of which can be relied upon to give the salient information for a given situation. Members of the agent community have recognised that in can be important for mobile code to be aware of the state of available network resources [9]; we believe, though, that such *network awareness* is potentially even more important for human administrators.

The importance of understanding network traffic has been recognised by others. The U. S. Department of Homeland Security rephrases our question into the term *situational awareness*, which it defines as "the ability to identify, process, and comprehend the critical elements of information about what is happening to the team with regards to the mission." [22] The DHS definition can be generalised to "knowing what is going on around you." [18] We use the term *network awareness* to refer to situational awareness applied to the area of computer networking, which we will define as "knowing what is happening on the network."

Most tools for monitoring high-bandwidth network connections analyse netflow records (such as those described in the proceedings of VizSec 2004 [30, 22, 6, 24, 16]), concentrating on traffic source and destination fields. Limiting enquiry to a subset of packet

headers makes sense for scalability reasons: at high data rates, packet payloads cannot be processed in a timely manner with commodity hardware. However, discarding packet payloads limits the scope of information available to administrators and researchers. Without the ability to analyze packet payloads, it is impossible to ascertain precise knowledge of the data crossing the network. To truly understand what is happening on a network, however, we need more than simple payload information. We must be able to reconstruct IP packets and TCP streams to provide our analysis tools with the same information available to the hosts at the endpoints of communications. We must then analyze the reconstructed data in a manner that allows us to detect what is happening in high level protocols, so that we can assign "intent" to network events, accurately saying why an event took place.

Thus, a tool that supports network awareness will allow network packet headers and payloads to be analysed and aggregated efficiently using an easy-to-use, responsive interface. As explained elsewhere, no such tool currently exists in the open source world. As a first step towards developing such a tool, we have designed and implemented qcap, a library for efficiently reconstructing network packets and streams, as well as analysing application level protocols. In the future, we plan to use qcap to develop user-friendly tools for understanding network data. Although qcap is not fast enough to analyze high-bandwidth data in real time, it is efficient enough to enable fast interaction with multi-gigabyte network captures using commodity hardware.

In our recent work on mitigating network denial of service [5], we have been studying strategies for automatically constructing packet aggregates. qcap was inspired by the lack of tools for analysing network content, and providing and explanation of network events. Because of the general need for better understanding of network behaviour, however, qcap should have a much wider appeal.

The rest of the paper proceeds as follows. In subsequent sections, we refine and explore the concept of network awareness; we provide an overview of existing tools that provide some degree of network awareness; we explore the possibilities network awareness offers us. We then present our work on qcap, a library for network awareness. We conclude with a discussion of limitations, challenges, and plans for future work.

Network Awareness

Network awareness is the ability to answer questions quickly and accurately about network behaviour. It is a well-developed understanding of a particular computer network that allows a system administrator (for example) to easily explain its behaviour, allowing the administrator to make rapid, well informed decisions. Because of the vast amounts of data involved, we cannot expect the administrator to be aware of every passing bit, but we can expect them to understand what classes of data to expect, the usual sources and destinations for most traffic types, and the identities of the users involved. Under normal network conditions, we expect that an administrator should be able to make reasonable accurate predictions about the network state in the near future. Under abnormal conditions, the administrator should be able to quickly quantify and describe the abnormality.

To help define the scope of network awareness, we present a series of questions whose answers provide some improvement in network awareness. While these questions are not comprehensive, they provide an indication of the kinds of issues that we might wish to understand but that are difficult to answer using currently available tools.

- **Who is using the network?** Many different entities use a network. We want to be able to determine who uses the network, and how they are using it. We are interested in different granularities of "user": applications, hosts, people, and other networks.
- **How is a host using the network?** We should be able to determine the services a host is providing and utilising. Services may run on non-standard ports, and may be tunnelled through other protocols.
- **How do different network events relate to each other?** Many network events occur as part of a larger chain of events. We can use an HTTP connection to illustrate: it begins with a DNS request, followed by an ARP (requesting the MAC address returned by the name server), followed by a TCP/IP connection to the named IP address. Finally, one or more HTTP requests are made to the requested web server. All of these events are causally related, and should be grouped together. Conceptually, we could go so far as to say that they are part of a single action.
- **How do low level protocols behave while being used by high level protocols?** We should be able to gauge how IP and TCP react to higher level payloads, such as SMTP.
- **What network traffic is encrypted?** It is normal for TLS and ssh traffic to be encrypted; other encrypted traffic on the network, however, could be evidence of attackers who are trying to conceal their actions.
- **What content is the network carrying?** In order to understand network use, some inkling of user-level intent should be available. The closest we can get to judging intent via the network is by attempting to reconstruct the human-level activities that the network is being used for. As such, we need to be able to use user-level concepts where necessary, such as emails, print jobs, and uploads.
- **What credentials are being used on the network?** Individual users may be associated with

multiple credentials. Where possible, we should be able to associate credentials with users. In environments where administrators have access to the encryption keys of users, it should be possible to decrypt passing traffic for further analysis.

- **What is the TCP state of all existing TCP streams?** Stream state can be an indicator of malicious activity: many new streams, unclosed streams, or streams that are timing out may be indicative of abnormal behaviour.
- **What is the meaning of some bytes in a specific stream?** Given the existence of signature-based intrusion detection schemes [4], it may be useful to put signatures into a context by reconstructing the streams around the signature match, to better understand the significance of the region.
- **What classes of interactions exist on the network?** Protocols can carry almost any type of data. Interactions should not be classified solely by the protocol(s) used, but by the content carried. For example, when dealing with emails that are being sent or received, it would make sense for POP, IMAP, and SMTP to be grouped together. However, some HTTP connections also carry email: so it would make sense for HTTP connections to web-based mail providers (such as Hotmail, and Yahoo) to be placed in the same class.

Note that these questions can only be answered through analysis of complete packets (headers and payloads), and that such analysis requires the aggregation of packets using syntactic, semantic, and temporal criteria. In particular, we will need to reconstruct packet streams (e.g., TCP streams, UDP-based multimedia traffic) in order to determine context and meaning.

While a detailed analysis of the current state of a network can help answer specific questions, the complexity of even small networks make such analysis difficult to comprehend for even the most skilled administrator. To accommodate this complexity, network awareness requires one to understand how network behaviour has changed over time. Because many changes are benign, we wish to know what looks anomalous about the current state relative to the past state. While anomaly detection in general is a difficult problem, the ability to classify packets more accurately should facilitate the development of improved network anomaly detection methods.

Existing Infrastructure

Although the questions described in the previous section are straightforward, current tools provide only limited support for answering these types of network awareness questions. Most of the tools described below were not designed as network awareness tools but have been pressed into service because of a lack of alternatives. Our tour of tools will start with libraries and work up to full applications. Note that this list is not intended to be exhaustive so much as illustrative: it provides examples of classes of application, not every application in each class.

The root of many existing packet capture tools is the BSD Packet Filter [23], which defines an elegant approach for winnowing packets based upon a textual predicate. The predicate is compiled into instructions for a tiny virtual machine, which can run in the packet capture device driver within the kernel. The BPF architecture has been widely accepted and incorporated into the libpcap [28] packet capture library.

In turn, libpcap and its parent application, tcpdump, have spawned a number of command-line based open-source progeny, including tcpstat [19] and tcptrace [1], that are well suited to auditing and debugging network traffic. In the more complex niches, we find ChaosReader [17], a command-line based stream reconstruction tool that is able to rebuild application level streams and store them as files; Snort [4], a signature-based intrusion detection tool; and Ethereal [2], a graphical packet display tool. These tools are designed primarily as network debugging and intrusion detection systems. Other tools [11, 10, 20] are useful for monitoring networked devices, diagnosing faults, and other administrative tasks. While each of these applications is well suited to locating known events, they are not well suited to providing information about the general state of the network.

The next level of abstraction to consider are network awareness tools. These are explicitly designed to provide network administrators with some sort of picture of the state of the network, usually for security purposes. An entire crop of these tools were presented at VizSec 2004 [22, 30, 6, 24], although older tools exist as well [7]. In general, most of these tools are graphical and use some variant of a two or three dimensional display to render network activity. The displays usually place network endpoints on two axes and the volume of traffic traveling between those endpoints on the third axis. Most of these tools present their output as a scatterplot [22, 6, 24], although dual axes graphs were also used [30].

For the most part, the VizSec 2004 tools provide statistical information on data traveling between source and destination endpoints. The endpoints may take the form of one or more networks, hosts, or ports. The data may be presented in terms of packets, connections, bytes, or some other volumetric measurement. Although volumetric data gives some indication of who is talking to whom, it does provide an indication of what is being said. Volumetric analysis does not provide answers to the questions we listed. We assert that volumetric analysis is insufficient to provide network awareness.

Forensic tools [3, 11] delve into the contents of data streams. They provide reconstructions of data

stream contents, often indexing it for searching, and support retrieval of text deemed to be of interest to users. They provide some idea of the classes of information that can be detected with full packet analysis. In particular, these tools can normalise network events into conceptual events and display those conceptual events grouped by type. For example, a listing of all discrete "login" events for the network can be presented, indexed by the credential used, and the resource acquired; as can all downloads (via HTTP, FTP, or BitTorrent); message sends (via SMTP, IM, SMS, or IRC); or file access (via SAMBA, NFS, or DAV). They can also provide access to the payload of application layer streams using an appropriate renderer (such as reconstructing and playing the audio portion of VoIP traffic).

While forensic tools might appear to be ideal tools for network awareness, their list-oriented interfaces are biased towards answering specific (not general) questions, provide little support for correlating high-level semantics with low-level packet behaviour, and offer few mechanisms for comparisons and anomaly detection. These limitations arise because these tools are designed to help dissect the specifics surrounding a particular incident rather than to help detect patterns that are not known in advance.

Interestingly, Q1 Labs QRadar [21] already provides a sophisticated network awareness tool. Although it provides many features that we are interested in and provides a mechanism to answer many of the questions listed above, we still feel that qcap is necessary. Although QRadar provides many features for real-time network awareness, it does not appear to provide sophisticated visualizations, a low-level API for traffic analysis, or provide mechanisms for performing automated analysis of stream content. These are not flaws in the product, per se, but are indications of Q1 Labs target market of security officers in large corporations. In contrast, qcap is aimed at the broader community of systems administrators and researchers who need to develop automated systems and custom visualization tools for studying network traffic.

The Promise of Network Awareness

To better understand the kind of applications we envision for qcap, here we present several visualisation idioms that would provide network awareness by quickly informing a network observer of the state of the network. There are clearly a huge number of other visualisation idioms, each well suited to some class of information; thus, this list is illustrative, not exhaustive. qcap provides the basic operations and abstractions that would be required to implement these visualisations efficiently.

Protocol State vs. Time

The intent of protocol state versus time display is to show how the state of two (or more) hosts change over time. Figure 1 shows a sample protocol state relative to time.

Figure 1 is ordered chronologically from top to bottom. Each side of the graph represents one of the hosts involved in the conversation. The left side is the initiator of the TCP connection, with IP address 192.168.0.1. The right side is the server 10.0.1.1; here the term "server" is used solely to denote that 10.0.1.1 was not the initiator of the connection. 192.168.0.1 is responsible for the first two packets sent, indicated with a right-pointing arrow angled down. Each arrow indicates an IP packet sent from one host to another; the height of the arrow indicates the amount of data the packet is carrying.

The meaningful data carried from one host to another causes the protocol to change states. Each state is global to the protocol and shown as a coloured rectangle. The rectangle encompasses all packets that

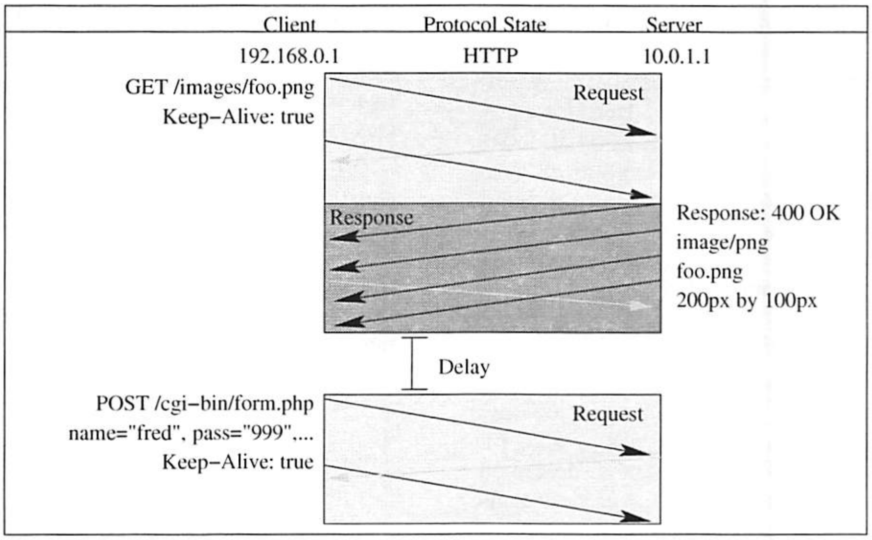


Figure 1: Display of protocol state relative to time from HTTP view.

are sent and received while the protocol is in the given state. Each state is coloured according to type, to allow the user to visually group the states.

Packets may carry information from multiple states. In other words, a hypothetical packet may contain three bytes, the first pushes the protocol into State1, the second pushes the protocol into State2, and the third pushes the protocol into State3. The arrow representing the packet would be shown as passing through State1, State2, and State3 in order.

Some of the packets exchanged on behalf of lower level protocols are meaningless to higher level protocols, so they are displayed in a different colour. Such packets include the TCP ACKs in Illustration 1, which are lightly greyed out, indicating that that do not carry any useful information for this protocol layer.

The display is annotated with blocks of text to the right and left of the state boxes. Each annotation supplies extra information about the state. Information supplied by a host is shown under that host. This means that the client sent a GET, which the server replied to with an image; later, the client sent a POST.

The protocol state at the top of the graph is intended to function as a combo box, allowing the user to select which level of the protocol stack they wish to view. Illustration 1 would therefore offer a selection between Ethernet, IP, TCP, or HTTP. If the interaction were part of a larger SOAP conversation, the SOAP view would be available as well.

This visualisation type is already provided by some existing commercial tools [10, 20].

Volume vs. Time

Figure 2 shows a traffic frequency graph that maps network activity to time. The x-axis is time, with the right-most portion of the graph being the most recent, and the leftmost being the oldest. The y-axis represents some value that changes over time. Like the host/aggregate vs. time graph, the traffic frequency graph can display one of many different y types (packet volume, traffic volume, average packet size, percentages of some total, or some attribute of a conversation that changes over time). The entities graphed are ordered vertically from least to most and named.

The graph is intended to be modal. The user should be able to switch the aggregates being graphed from protocols (shown) to hosts, groupings of hosts, or any grouping of conversations. In addition, the user should be able to drill down into a class of traffic, to display on that with finer granularity divisions.

Aggregate State vs. Time

The state of a host can be viewed by watching its actions change over time. Figure 3 displays permutations of the host/aggregate against time. Although any variable could be displayed on the vertical axis, for ease of explanation, we shall assume that the variable being displayed is that of packet volume.

The graph is divided into five rows. Each row is a different mode of display that provides a summary of information about the entity on the left side of the row. The entity is shown as a glyph (either a single host in rows 1 to 4, or a subnet in row 5). Beside the glyph is a listing of protocols the entity is using. To the right of the protocol list, the state vs. time listing is shown. The graph displays information about the network against time. Note that all time-based graphs scroll from right to left: the rightmost data is the most recent.

The first row shows an aggregation of values for a host. The host, 192.168.1.2, is producing a volume of data for three listed protocols (HTTP, POP, and SSH). The graphs are summaries for all conversations that the host is participating in. In other words, it may be involved in many POP connections, but the value shown is the total for all POP connections. As the "+" to the left of the protocol name suggests, there is a hierarchical listing of information that the user may gain access to by "opening" that protocol.

Since there are many ways of grouping aggregated information together, rows 2 and 3 will illustrate three possible forms of grouping. It is our intent that the "+" beside the protocol name be a means of cycling through the three possible views.

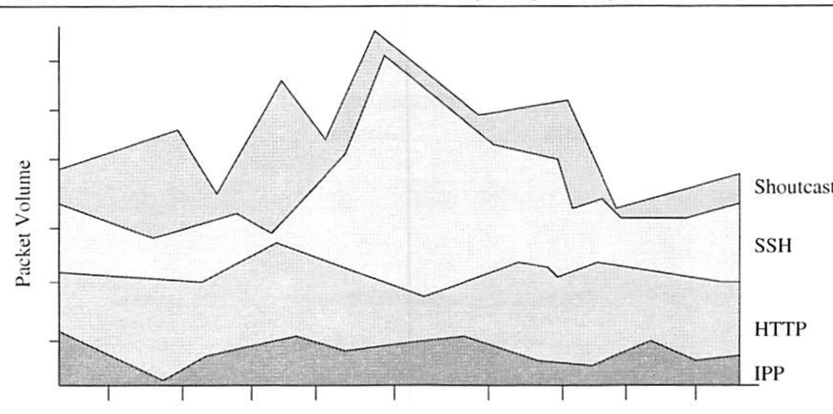


Figure 2: Display of traffic volume against time.

Row 2 provides a brief textual description of the state of each protocol the host is using. The descriptions are protocol dependent. The third row features a detailed summary of the HTTP traffic of 10.0.0.1. The only two items available for HTTP are the total number of connections, as well as the total number of connections to unique hosts. The values displayed depend on the protocol being viewed: they could easily include the number of web pages downloaded; the average number of objects per web page; the hosts currently connected to; or the user agent performing the requests.

The fourth row shows a breakdown of conversations that the host 10.0.1.2 is participating in. It is one of the modes mentioned under "Row 3."

- The top line, labelled "HTTP," shows a graph of the total volume of traffic for the given protocol type; this total is the total for all of the conversations listed underneath it.
- The second line shows a graph of the volume of traffic for the conversation 10.0.1.2 is having with 20.2.2.2.
- The third line shows the conversation that 10.0.1.2 is having with 30.3.3.3 as a listing of protocol states. Each state is concisely named, and tagged with a protocol-specific colour. If the state is too thin to display a name, it will simply be coloured. Because the rightmost side is the "newest" side, the states will be described in an English-readable manner.

The fifth row shows a grouping of hosts on the network 192.168.*.*. Like the per-host display, each protocol is displayed as a graph. The protocol can be opened to display either a textual summary or a listing of all conversations it contains.

With the aid of a context menu, the user could add or remove hosts from this aggregate.

Summary

The three visualisations described in this section require features not present in known open source libraries and tools. They include:

- **Indexing** provides fast access to network information based upon queries, or walking displays. This allows the graphical display to be recomputed quickly, by processing only those packets that are relevant to the current visualisation.
- **Random access into packet traces** must be provided to allow the visualisation tool to properly exploit the features of indexing. Even if the tool is designed to run online, it will have to keep either a rolling buffer of recent data if any kind of historic display is to be provided.

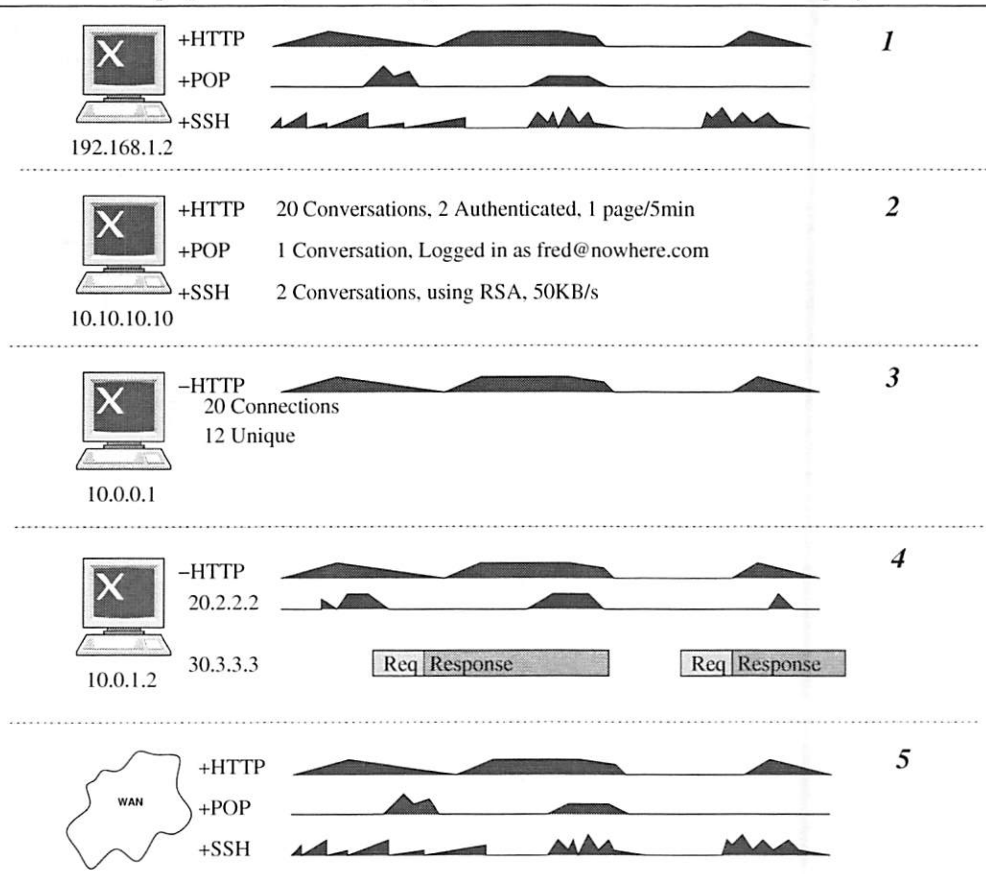


Figure 3: Display of aggregate state against time. Each row indicates the state of a specific entity.

"Recent data" in this context can refer to packets or aggregates of traffic information that are stored in memory.

- **Task-specific internal models** allow large data sets to be aggregated, manipulated, and displayed quickly and efficiently. Such models also provide the means to create very informative and expressive slaved visualisations [26].
- **Access to network, transportation, and application layer data** is necessary to provide a high level view of data.

More complete discussions of the basics of visualisation are available in [27, 26].

A Library for Network Awareness: qcap

The features listed in the Summary cover a wide variety of computer science disciplines, from database maintenance and access to visualisation techniques, that have been thoroughly addressed elsewhere. However, one feature is currently missing from the pantheon of tools and libraries available: we have no tools for the wholesale decomposition of large volumes of packets in speeds approaching real-time. There are no known APIs or libraries for reconstructing conversations from packet traces, and subsequently decomposing those conversations into their logical parts.[1] We wish to gain access to the payload of packets to examine the relationships between application-layer payload, individual packets, transportation layer interactions, and network events. In order to do that, we need a means of reconstructing application-layer streams that preserves network layer information. The best means that we can see to do that is to provide a processing layer that sits on top of a BPF implementation and provide higher network reconstruction functionality.

We have created the qcap library to address these needs. It is an open source library that uses an IP and TCP reconstruction engine derived from libnids library, which in turn was derived from a version of the Linux 2.0 networking stack [29]. It can translate individual packets into complete conversations. Those conversations can be tapped at any point during reconstruction, to allow the tool-user to fully understand the significance of each packet. qcap provides:

- **Packet parsing** which allows an application to query fields in a packet. For example, given a domain name query packet, we would be able to query for any of the fields in it: be they IP, UDP, or DNS.
- **Packet reconstruction** rebuilds fragmented IP packets. Different network stacks can reconstruct corrupted or malicious fragmented packets in different ways [25]. qcap provides a mechanism to allow an application to control how fragmented packets are rebuilt, allowing qcap to properly emulate different network stacks.
- **Stream reconstruction** creates stream "objects" from a set of packets. The stream objects allow the application to read a datastream that is identical to that on the receiving end of the connection.
- **Stream parsing** provides a means for the application to easily dissect a conversation. qcap parses the stream text, allowing the application to request specific syntactically defined portions of that text.

Design of qcap

We use libpcap as a guide for our design. It is old, well established, and still actively maintained. It appears to be the most popular open source packet acquisition library, used in numerous open source projects [28, 19, 1, 4, 2, 29]. Its design is simple, offering a subscription interface to listen for packet arrival.

We provide two subscription interfaces, one for packets, and one for portions of TCP streams. The packet subscription function is qcap_packet_handler_add(), which associates the callback with a specific stage of packet reconstruction or stream assembly. Meanwhile, TCP streams can be subscribed to with qcap_tcpstr_handler_add(), which causes a callback to be triggered when a specified syntactic element in an application-level stream is found.

To support these two functions, we have three types of object: qcap_packet_t, qcap_tcpstr_t, and qcap_tcpstr_pos_t.

Packets

Network level objects are instances of qcap_packet_t's. Because our network level provides packet reconstruction for fragmented IP packets and an indication of logical events each packet generates, we add two flags to each packet: the *artificial* flag and the *discarded* flag. The artificial flag is used to denote defragmented IP packets, while the discarded flag is used to denote packets that are known to have been dropped.

Our packet abstraction provides the following information:

- **Text** is the data sent across the network layer. It is used for reconstruction by higher layers.
- **Arrival time** when the packet was received at the sampling point.
- **Discarded** is a flag that indicates whether or not the packet has been dropped before final delivery, either due to network state, or the reconstruction policy of the recipient endpoint.
- **Artificial** is a flag that indicates if the packet was constructed artificially from other packets.
- **Constituents** is a list of packets that this packet was built out of. Only artificial packets have constituents.
- **Fragment** is a flag indicating that this packet is a part of another packet.
- **Processing state** is the stage of processing that the packet is in. There are many stages, they are

[1] libnids [29] performs stream reconstruction, but does not provide packet or stream decomposition.

used to indicate how the destination network stack is expected to treat the packet. Possibilities include: if a packet was discarded due to some logic error (such as a failed IP CRC check), if the packet triggered the creation of a new TCP connection, or if it is being queued due to TCP ordering issues.

Each packet passes through many states as it is processed. The application may subscribe to packets entering any state.

TCP Streams

A TCP stream (or qcap_tcpstr_t) is an ordered collection of qcap_packet_t's, in proper TCP order. qcap_tcpstr_t is an opaque data type, but can be queried with stream positions (or qcap_tcpstr_pos_t).

TCP Stream Positions

Stream positions are positions at exact locations within a stream. They are implemented as an opaque data type that can be copied, walked forward in the stream, and have the byte at their location queried. Given two positions in the same stream, the application can request all of the bytes between the positions.

Because the qcap_tcpstr_t type is opaque and only queryable through qcap_tcpstr_pos_t types, it allows qcap to perform reference counting and garbage collection on individual packets within a qcap_tcpstr_t.

Analysing Fields in Packets

In addition to the callback interfaces described above, qcap also provides a mechanism to query fields from packets. In protocols that have primarily fixed-length fields, such as IP, TCP, and UDP headers, querying fields is trivial: it only requires byte-order conversion and a cast to a native type. However, other protocols such as DNS have arbitrary-length fields and non-standard data encoding, meaning that an application-writer must write complex code to perform a simple task.

Field querying is done with the qcap_getter_t type. A call to qcap_getter_compile() creates a "getter" from a string specification. Packets can be queried by calling qcap_getter_apply(). The result is converted into a particular form, such as a string, or a boolean, and returned.

Implementation

qcap is written in C and is built on top of libpcap. Although heavily modified and extended, parts of pcap are also derived from libnids. At this stage, pcap does not use any other utilities outside of the standard platform libraries. As the bulk of the newly written code deals with parsing stream-based protocols, we shall discuss that code here.

One of the non-functional requirements surrounding qcap was that it must be easy to extend with new protocols. Towards this end, qcap internally uses context-free grammars that are generated from static C code. As it turns out, however, many protocols cannot practically be defined solely with a context-free grammar, as the protocol carries information about its own syntax. Such information is carried in a field providing a parameter, such as the length of a subsequent field, or a field that provides a terminating delimiter for some subsequent field.

For example, consider HTTP. Well formed HTTP streams consist of requests (originating with the client), and responses (originating with the server). Both requests and responses can carry an optional "body" that can have an arbitrary length. The length of the body is usually defined with a "Content-Length" field that contains an integer indicating the number of bytes in the body. We could express the "Content-Length" statically in the HTTP context-free grammar, by enumerating every possible value of "Content-Length", and defining the length of the subsequent body in the context-free grammar. However, that would result in an extremely verbose grammar.

To avoid such large grammars, qcap implements a series of protocol-specific registers for each parser. As the parser is walking a stream, it places the value of specific stream elements into the associated register, which can be used later during the parse. Continuing with our example from above, qcap would store HTTP's Content-Length field with a content_length register in the stream parser. Upon encountering the Content-Length field, the parser would decode the associated value as an integer and store it in the content_length register. Later, when the parser encounters the subsequent body, it will read exactly content_length characters.

While current grammars are built by hand, we are researching mechanisms for automatically generating the necessary static C code from protocol specifications in Augmented Backus-Naur Form (ABNF) [12].

Evaluation

In our opinion, the most interesting feature of qcap is the analysis of application-layer protocols. We believe that qcap will be most useful if it is able to perform application-layer analysis at speeds approaching real-time.

We have developed two test applications to test qcap's speed:

- reader opens a trace and parses it, performing IP defragmentation and TCP stream reconstruction. It provides no useful output and is used for timing purposes.
- ip_identity opens a trace, and gathers credentials sent from each IP address in the trace. Currently, ip_identity only parses FTP usernames and passwords, SMTP senders, and HTTP authorisation requests [14].

Both ip_identity and valid reconstruct all IP fragments and TCP streams. qcap could be set to ignore TCP/IP traffic going to ports that we aren't interested in, but these tests provide us with a stronger worst-case idea of processing time.

To test timing, we ran each program against the Lincoln Laboratory DARPA Intrusion Detection Evaluation datasets [32]. The first five datasets are DIDE-1 to DIDE-5, which correspond to the Monday-Friday traffic of the LBL week 1 traffic set, DIDE-6 to DIDE-10 correspond to the data gathered during week 3. Each represents one day's worth of traffic.

The tests were run on an unloaded 2.8 Ghz Pentium 4 system with 1 GB of RAM, 512K of processor cache, and two 36.4 Gb, 10,000 RPM SCSI drives. Each test was run 1000 times, and the values were averaged.

In order to minimise the amount of time spent on output, we ran tcpstat with an invalid BPF filter, ensuring that it would not waste too much time writing data. Since reader does not produce output, and ip_identity only produces brief output at the end of the trace, we did not modify either application to reduce their volume of output.

During our experiments, we discovered that qcap analyses data at a rate of roughly 2.11 microseconds per packet, as compared to tcpstat which analysed traffic data at a rate of 0.694 microseconds per packet. The difference in processing speed amounts to an order of magnitude: however, as shown in Table 4, processing times are still low: to process a 468 MB trace file (DIDE-6) containing 2.1 million packets with qcap only takes about 5.7 seconds.

Since ethereal seems to be the flagship open source packet processing tool, we also timed how long it took ethereal to open each packet trace. In our experience, this provides a ballpark figure on how long it takes ethereal to perform a search. These tests were performed a handful of times on the same machine mentioned above, with the lowest timing result provided. As shown in Table 4, ethereal is between 3 and 20 times slower than qcap.

In our judgment, the rates achieved by qcap are acceptable: we can analyze a 1 GB trace in roughly 10 seconds. As we increase the number of protocols that we are parsing and the complexity of the protocol parsers, this execution time will increase; however, we expect the parse time to stay within the same order of magnitude. In addition, the current implementation of qcap has not undergone any optimisation, suggesting that we may be able to achieve speed improvements with minimal effort.

Discussion

At this point, skeptical readers may be asking themselves what is new about qcap: tools already exist for analysing packet traces. Indeed, any of the information we acquire with qcap can also be acquired by using existing tools. For example, if a user wants to parse all of the cookie headers out of an HTTP conversation, they could use ngrep with a specially constructed regular expression. Or, if a user wanted to find the contents of a TCP session, they could use tcpflow to pull all of the TCP sessions out of a trace, and then analyze them by hand or open them with ethereal. And general connection statistics can be gathered with much simpler tools, like tcpdump or tcptrace. Alternatively, snort could be used for any of the aforementioned tasks.

Those skeptical readers should realize that qcap has been designed specifically for network awareness. It is not designed to find individual packets in a network trace, nor is it designed to find statistics on a specific class of event. Instead it is designed to:

- Provide a standard interface to analyze traffic, regardless of protocol.
- Perform full stream reconstruction, allowing the application to parse strings that are split across multiple TCP packets, without having to be aware of the packet divisions. Applications can, however, request the information to be made available to them.
- Perform the drudge work of protocol syntax analysis, allowing the application to concentrate on the meaning of the traffic.
- Provide a simple mechanism for collecting a wide variety of data and statistics.
- Handle large volumes of data quickly.

To our knowledge, no other open-source library provides this functionality. We have not seen an open-source tool that correlates large volumes of application-level network data.

File		**Processing Time (seconds)**				**Time Per Packet (microseconds)**			
Test	*Packets*	tcpstat	ip_identity	reader	ethereal	tcpstat	ip_identity	reader	ethereal
DIDE-1	1,362,869	0.74	4.48	2.93	69	0.54	3.29	2.15	50.6
DIDE-2	1,157,328	0.68	4.35	2.65	60	0.58	3.76	2.29	51.8
DIDE-3	1,616,713	0.87	5.23	3.43	86	0.54	3.24	2.12	53.2
DIDE-4	1,807,060	1.07	6.32	4.09	94	0.59	3.50	2.26	52.0
DIDE-5	1,349,635	0.70	4.25	2.82	70	0.52	3.14	2.09	51.9
DIDE-6	2,106,744	1.12	5.67	3.94	109	0.53	2.69	1.87	51.7
DIDE-7	1,831,648	0.97	5.45	3.48	98	0.53	2.97	1.90	53.5
DIDE-8	1,849,753	1.13	6.97	4.35	113	0.61	3.77	2.35	61.1
DIDE-9	1,559,156	0.74	3.38	2.65	85	0.48	2.17	1.70	54.5
DIDE-10	1,635,425	1.01	6.67	3.93	101	0.62	4.08	2.40	61.8

Figure 4: Observed processing speeds of qcap compared to those of tcpstat.

Limitations

While we believe qcap is quite promising, it is also a new library with many limitations. For example, because qcap builds upon libpcap, we can, for the most part, say that qcap shares a subset of libpcap's limitations. There are a number of other limitations, however, that are specific to pcap.

First, qcap is not suited to searching for known strings in input text, either as a literal string, or a regular expression. Specialised tools, such as ngrep perform those tasks well. However, since qcap is a thin wrapper around libpcapthere is no reason why such tools could not be ported to use qcap.

Next, even though qcap's protocol parsing capabilities are well-developed, it does not deal with lookahead. Lookahead means scanning further in the stream of text being parsed to discover if any text in the near future would prevent the current text from being deemed valid.

A trivial example involves HTTP requests. A valid HTTP request consists of the request-line, such as a "GET <url> <protocol>". When qcap is parsing the request-line, and it reaches the end of the *url* field, it will inform the application that a *url* field has been encountered and then continue parsing with the *protocol* field. If the *protocol* field turns out to be invalid (because it contains the text "foo" instead of "HTTP/1.1," for example), then the entire *request-line* should be deemed invalid, meaning that qcap should not of informed the application that an *url* was discovered earlier.

The difficulty with lookahead is that it involves reading data into memory before deciding if a chunk of text is a part of a valid stream. If we need to perform lookahead to the end of a large piece of data, we could flood memory with data: consider an HTTP response that contains a 50 MB file – the response headers cannot be considered valid until the entire response is received, but that means that qcap would have to read the full 50 MB file into memory before deciding if it should accept or reject the response headers. Instead, we force the application developer to be aware of the protocol structure, and watch for failed requests.

qcap, because of limitations in libnids, does not currently support any kind of parameterisation to state how streams should be rebuilt in circumstance not defined by the IP and TCP RFCs. Such situations include packets with invalid CRCs, overlapping packets, et cetera.

One potential issue for some applications is that qcap currently has no mechanism for synchronizing the two sides of a stream during parsing. Strictly speaking, such synchronisation is necessary to ensure that proper protocol state is maintained at all times. However, in practice we have not yet found the lack of synchronisation tracking to be a problem.

However, one of the features we feel will be necessary for any offline analysis GUI tool built on top of qcap is analysis of network traces larger than one gigabyte. In order for such an analysis to be relatively fast, and put a low load on the workstation, a minimum of data should be kept in memory; suggesting that, where possible, information about specific packets should be kept on disk, and read as necessary. To speed up overall performance and searching, we assume that indices would be built during the initial load.

In order for packet data to be left out of memory, we need a means to provide random access to a static trace file on disk – meaning that we should be able to read packets from the file in an arbitrary order. However, libpcap does not support random access into trace files, meaning that solely reading specific packets from disk is not possible.

There are two possible approaches to this problem: either petitioning the libpcap maintainers to include random-access to libpcap files; or building our own file reading routines into qcap. Clearly, the first option is preferable. At the time of writing, we are engaged in petitioning the libpcap maintainers to include this functionality.

Future Work

While internal parts of the library are still evolving, at the time of writing, the qcap API is almost complete. The existing functions are unlikely to change for the foreseeable future, even though new calls may be added. Having said this, there are are some issues that we hope to address in the near future.

First, there are currently no mechanisms for decoding stream content. qcap should be able to decode either an entire stream (such as ssh or SSL), or portions of a stream (such as gzip-encoded HTML responses), in a manner that is transparent to the application. It should be possible for encoded regions to contain semantic elements that are to be recognised by stream parsers. Such additions would significantly improve the utility of qcap.

We also plan to develop bindings to allow higher level languages such as Python, Perl, and Java to access qcap functionality. Such changes and additions should help facilitate the development of novel network awareness applications. The qcap distribution contains sample programs that provide interesting functionality not seen in other open source tools: ip_identity trawls network traces for credentials; and valid tests streams to see if they follow the protocol semantics for the ports they are using. While such tools can be useful, much larger scale applications are also possible:

- a fast protocol debugger, along the lines of ethereal, but supported by a database back-end that would provide fast searching and display.
- anomaly-detection tools that consider the values of individual fields in a protocol stream.

- "leak" detection tools that sniff passing traffic for sensitive content that should never leave hosts.
- a fast classification tool that classifies streams by their purpose, either in gross terms; such as "exchanging email" for SMTP, POP, IMAP, Gmail, and Hotmail connections; or specific terms, such as "instant messenger conversation between Alice and Bob."

In the long term, optimisation and improvement of qcap will allow it to process extremely high volumes of data. Ideally, qcap will eventually be able to handle data at rates approaching those seen by medium-to-large ISPs and enterprises. When it does, our definition of network awareness can grow from today's analysis of traffic volumes to and from hosts to include content-specific and aggregate analysis that will finally help us figure out what our networks are actually doing.

Acknowledgements and Availability

This work was supported by the Canadian government through an NSERC Discovery Grant and MITACS. The qcap library can be downloaded from http://www.ccsl.carleton.ca/projects/qcap. It is licenced under the GNU General Public License (GPL).

About the Authors

Evan Hughes graduated from Carleton University in Ottawa, Canada in 2000 with a BCS. Since then he has worked for a number of start-up companies in the software space; before retiring from the 9-5 world to do good works contracting with charities. Given his fondness for food and shelter, he quit contracting for the penniless and has enrolled at Carleton University for his MCS. He is currently doing research work in the Carleton Computer Security Lab. Email reaches him at evan.c.hughes@gmail.com .

Anil Somayaji is an assistant professor in the School of Computer Science at Carleton University and is associate director of the Carleton Computer Security Laboratory. His research interests include operating system security, intrusion detection, complex adaptive systems, and artificial life. He received a B.S. in Mathematics from the Massachusetts Insitute of Technology in 1994 and a Ph.D. in Computer Science from the University of New Mexico in 2002. He can be reached at soma@ccsl.carleton.ca.

Bibliography

[1] *tcptrace homepage*, http://www.tcptrace.org , Accessed May 3, 2005.

[2] *Ethereal homepage*, http://www.ethereal.com , Accessed May 3, 2005.

[3] *Netwitness homepage*, http://www.netwitness.com , Accessed May 3, 2005.

[4] *snort homepage*, http://www.snort.org/ , Accessed: May 3, 2005.

[5] Matrawy, Ashraf, Paul C. van Oorschot, Anil Somayaji, "Mitigating network denial-of-service through diversity-based traffic management," *Applied Cryptography and Network Security (ACNS) 2005*, pp. 104-121, 2005.

[6] Ball, Robert, Glenn A. Fink, and Chris North, "Home-centric visualization of network traffic for security administration," *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pp. 55-64, ACM Press, New York, NY, 2004.

[7] Becker, Richard A., Stephen G. Eick, and Allan R. Wilks, "Visualizing network data," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, Num. 1, pp. 16-28, 1995.

[8] Bellovin, Steven M., "Packets found on an internet," *SIGCOMM Comput. Commun. Rev.*, Vol. 23, Num. 3, pp. 26-31, 1993.

[9] Wilmer Caripe, et al., "Network awareness and mobile agent systems," *IEEE Communications Magazine*, July, 1998.

[10] *Clearsight*, Clearsight analyzer homepage, http://www.clearsightnet.com/products-analyzer.jsp , Accessed May 4, 2005.

[11] *Colasoft*, Colasoft capsa, http://www.colasoft.com/products/capsa.php , Accessed May 3, 2005.

[12] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF," *RFC 2234*, November, 1997.

[13] Endsley, Mica R. and Daniel J. Garland, editors, *Situation Awareness Analysis and Measurement*, Lawrence Erlbaum Associates, 2000.

[14] Franks, J., P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication," *RFC 2617*, November, 1997.

[15] Gates, Carrie, Michael Collins, Michael Duggan, Andrew Kompanek, and Mark Thomas, "More netflow tools: For performance and security," *18th Large Installation System Administration Conference (LISA '04)*, pp. 121-132, Atlanta, Georgia, November, 2004.

[16] Goldring, Tom, "Scatter (and other) plots for visualizing user profiling data and network traffic," *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pp. 119-123, ACM Press, New York, NY, 2004.

[17] Gregg, Brendan, *Chaosreader homepage*, http://users.tpg.com.au/bdgcvb/chaosreader.html , Accessed May 3, 2005.

[18] U. S. Coast Guard, *Team coordination training student guide*, http://www.cgaux.info/g_ocx/training/tct , Accessed May 4, 2004.

[19] Herman, Paul, *tcpstat homepage*, http://www.frenchfries.net/paul/tcpstat/ , Accessed May 1, 2005.

[20] Network Instruments, *Observer homepage*, http://www.networkinstruments.com/products/observer.html, Accessed May 4, 2005.

[21] Q1 Labs, *Qradar product page*, http://www.q1labs.com/products/prod_overview.html, Accessed September 27, 2005.

[22] Lakkaraju, Kiran, William Yurcik, and Adam J. Lee, "NVisionIP: netflow visualizations of system state for security situational awareness," *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 65-72, ACM Press, New York, NY, 2004.

[23] McCanne, Steven and Van Jacobson, "The BSD packet filter: A new architecture for user-level packet capture," *Proceedings of the 1993 Winter USENIX Conference*, pp. 259-270, 1993.

[24] McPherson, Jonathan, Kwan-Liu Ma, Paul Krystosk, Tony Bartoletti, and Marvin Christensen, "Portvis: a tool for port-based detection of security events," *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pp. 73-81, ACM Press, New York, NY, 2004.

[25] Ptacek, Thomas H., and Timothy N. Newsham, *Insertion, evasion, and denial of service: Eluding network intrusion detection*, Technical report, Secure Networks, Inc., Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, 1998.

[26] Roberts, Jonathan C., "Multiple-View and Multiform Visualization," Robert Erbacher, Alex Pang, Craig Wittenbrink, and Jonathan Roberts, editors, *Visual Data Exploration and Analysis VII, Proceedings of SPIE*, Vol. 3960, pp. 176-185, IS&T and SPIE, January, 2000.

[27] Shneiderman, Ben, "The eyes have it: A task by data type taxonomy for information visualizations," Technical Report UMCP-CSD CS-TR-3665, University of Maryland Computer Science Department, 1996.

[28] *tcpdump workers*, Tcpdump public repository, http://www.tcpdump.org, Accessed September 27, 2005.

[29] Wojtczuk, Rafal, *libnids homepage*, http://libnids.sourceforge.net/, Accessed May 5, 2005.

[30] Yin, Xiaoxin, William Yurcik, Michael Treaster, Yifan Li, and Kiran Lakkaraju, "Visflowconnect: netflow visualizations of link relationships for security situational awareness," *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pp. 26-34, ACM Press, New York, NY, USA, 2004.

[31] Zalewski, Michal,*Museum of broken packets*, http://lcamtuf.coredump.cx/mobp/, Accessed May 6, 2005; 2003.

[32] Zissman, Marc, *DARPA Intrusion Detection Evaluation Datasets*, http://www.ll.mit.edu/IST/ideval/data/1999/1999_data_index.html, Accessed September 27, 2005; 1999.

Toward a Cost Model for System Administration

Alva L. Couch, Ning Wu, and Hengky Susanto – Tufts University

ABSTRACT

The core of system administration is to utilize a set of "best practices" that minimize cost and result in maximum value, but very little is known about the true cost of system administration. In this paper, we define the problem of determining the cost of system administration. For support organizations with fixed budgets, the dominant variant cost is the work and value lost due to time spent waiting for services. We study how to measure and analyze this cost through a variety of methods, including white-box and black-box analysis and discrete event simulation. Simple models of cost provide insight into why some practices cost more than expected, and why transitioning from one kind of practice to another is costly.

Introduction

What is a set of "best practices"? Ideally, it is a set of practices that cost the least while having the most value, i.e., a model of practice for which the ratio value/cost is maximal over the lifecycle of the equipment being managed. We have not succeeded in evaluating practices according to this yardstick, however, because there is no usable model of cost for any particular set of practices. We would like a model that predicts, based upon particular management decisions, the total cost of operations that results from those decisions over the lifecycle of the network being managed. This is one goal of "analytical or theoretical system administration" [5, 6].

Many system administrators and managers consider a complete cost model to be an impossible goal for several reasons. First, the actual cost of system administration is a relatively constant and monolithic line item in many IT budgets; it is commonly technically infeasible to break the lump sum cost into component costs for the purpose of evaluating strategy. Mechanisms for recording day-to-day costs (e.g., detailed time sheets) are often more expensive to manage than the money they might potentially save. And for the organizations whose audit requirements force them to maintain detailed costing data, these records are usually confidential and unavailable to researchers outside the organization. Thus any really usable cost model has to be practical in not consuming resources, tunable for specific situations by the end-user, and must allow that user to incorporate confidential data into the model without divulging it to outsiders.

Currently, instead of considering costs, we justify best practices by what might best be called a "micro-economic" model. We say that certain practices "make the job easier", or other weak justifications. Is "simpler" really "cheaper"? We have yet to prove this assertion and – in many cases – the microcosmic reasoning we use today seems to be incorrect at a macrocosmic (lifecycle) scale. A case in point is the use of automation, which is "better than manual changes" except that – at a macrocosmic scale – scaling increases costs in ways that are inconceivable when working on a small number of machines. The reasons for this apparent contradiction are deep and will be discussed later in the paper.

Current Ideas About Cost

The first step toward a cost model was made by Patterson [18], who points out that while "administrative costs" may be fixed and non-varying, the cost of downtime varies with scale of outage and disrupts more than computer use. Patterson's formula for the cost of downtime is based upon calculation of two factors we previously ignored as system administrators: revenue lost and work lost. Even if our system administration group has a fixed size and operating budget, the cost of downtime varies with the severity and scope of outage, and lifecycle cost of operations thus varies with risk of outage. Patterson also points out that there are more subtle costs to downtime, including morale and staff attrition. But how do we quantify these components in a cost model?

Cost modeling also depends upon risk and cost of potential catastrophes. Apthorpe [1] describes the mathematics of risk modeling for system administrators. By understanding risk, we can better make cost decisions; the lifecycle cost of an administrative strategy is the expected value of cost based upon a risk model, i.e., the sum of "cost of outcome" times "probability of outcome" over all possible outcomes.

Cost modeling of risks is not simple; Cowan, et al. [8] point out that simply and blindly mitigating risks does not lead to a lowest-cost model. It is often better to wait to apply a security patch rather than applying it when it is posted, because of the likelihood of downtime resulting from the patch itself. Thus the global minimum of lifecycle cost is not achieved by

simply minimizing perceived risks; other risks enter the system as a result of mitigating the perceived ones.

Further, Alva Couch made the bold claim at the last LISA [7] that the essential barrier to deployment of better configuration management is "cost of adoption". The reason that configuration management strategies such as automation are not applied more widely is that it costs too much to change from unautomated to automated management. But he stopped short of truly quantifying cost of adoption in that presentation, due to inadequate models. Meanwhile, many people pressured him in one way or another to formalize the model and demonstrate his claims rigorously. This paper is the first small result of that pressure.

In this paper, we make the first step toward a cost model for system administration, based upon related work in other disciplines. We begin by defining the components of an overall lifecycle cost model. We look at real data from a trouble-ticketing system to understand the qualities of load upon a support organization, and discuss the problems inherent in collecting data from real systems. We explore the relationship between system administration and capacity planning, and show that we must determine specific rates in order to determine costs. We borrow mechanisms for determining those rates from white-box and black-box cost models in software engineering. Finally, we turn to discrete event simulation in order to understand the relationships between rates and cost. As a result, we can begin to quantify the cost of some decisions about practice, including deployment schedules for new software.

A Simple Model Of System Administration

First, system administration can be modeled as a queueing system (Figure 1) in which incoming requests arrive, are queued for later processing, and eventually dequeued and acted upon, and completed. Each kind of request arrives at the queue with an "arrival rate" and is completed in a length of time whose reciprocal represents a "service rate." We embody all changes made to the network as requests; a request may indicate a problem or ask for a change in the nature of services offered. Requests arise from many sources, including users, management, and even the system administrator herself may make a note to change something. Likewise, requests are granted via many mechanisms, including work by system administrators and work by others.

Note that this is a more complex model than represented by the typical helpdesk. In a typical ticket system, tickets represent *external* requests, while *internal* requests (e.g., actions generated by a security incident report) are not given ticket numbers. In our request queue, all change actions are entered into the queue, serviced, and closed when done.

System administration has complex goals, so the request queue has a complex structure; it is (in the language of capacity planning [17]) a *multi-class queueing system* consisting of a mixed set of several "classes" of requests (Figure 2). Many kinds of requests, with different average arrival rates, are combined into one request stream. Each kind of request K

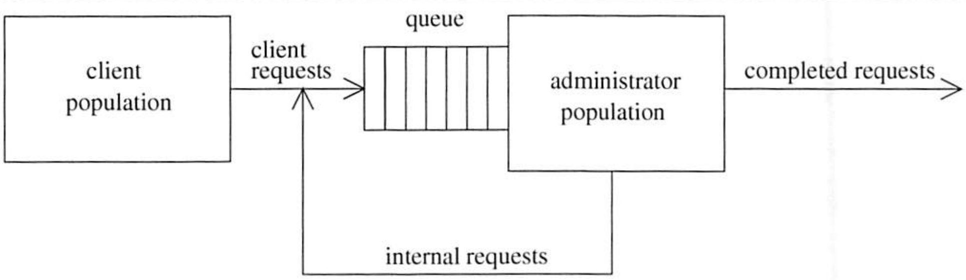


Figure 1: System administration as a queueing system.

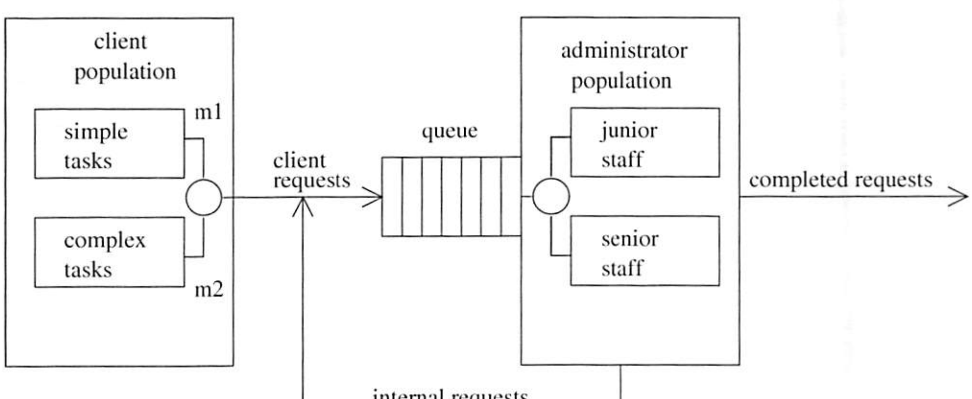


Figure 2: Multiple classes of requests and administrators.

has a distinct average service rate μ_K (and perhaps, a distinct statistical distribution of service times). As well, a realistic system administration organization is a *non-product system*: system administrators do not function independently like a set of cloned web-servers; they communicate and interact with one another, affecting throughput. A *product system* (as in Cartesian product) consists of a number of components that function independently (so that the state-space of the whole system is a Cartesian product of the state-spaces of its parts).

Request Arrivals

While the overall structure of the request queue is complex, we observe that the structure of some classes of requests is easy to understand. Many classes of requests arrive with a "Poisson distribution" of inter-arrival times. In a Poisson distribution with arrival rate of λ requests per unit time,

1. The mean inter-arrival time is $1/\lambda$.
2. The standard deviation of the inter-arrival time is $1/\lambda$.
3. The arrival process is *memoryless*; the probability that a request will arrive in the next t seconds is independent of whether one arrived recently.

Many kinds of requests naturally obey this distribution. For example, any kind of request in which a large population operates independently of one another has a Poisson distribution, e.g., forgotten passwords.

As well, many non-Poisson classes of requests (e.g., virus attacks) arrive with a Poisson distribution if viewed at the proper scale. While requests for virus cleaning of individual hosts arrive in bursts and are not memoryless, the arrival of the virus at one's site is an independent, memoryless event. If we treat the virus arrival at one's site as *one* event, rather than the thousands of requests it may generate, then new viruses arrive with a roughly Poisson distribution (because they originate from independent sources at relatively constant rates). Likewise, while an outage of a particularly busy server may generate thousands of tickets, the outage itself obeys a Poisson distribution even though the tickets resulting from the outage do not. Many other kinds of requests have this character; although actual tickets arrive in bursts, the real problem occurs in a memoryless way that is independent of all other problem occurrences. Examples include hardware failures, power outages, denial-of-service attacks, spam, etc.

Request Processing

The second part of our model is how requests are processed. Like request arrivals, request processing is complex but there are parts of it that are understandable. For example, many kinds of requests are completed via an "exponential distribution" of service time. The properties of an exponential service time are similar to those for a Poisson arrival; if a class of requests is serviced with an exponential rate of μ requests per unit time, then:

1. The mean time for servicing a request is $1/\mu$.
2. The standard deviation of service time is $1/\mu$.
3. The service process is *memoryless*; the probability that a request will be finished in the next t seconds is independent of whether we know it has been in progress for s seconds already.

The last assumption might be paraphrased "A watched pot never boils."

Examples of requests that exhibit an exponential service time include password resets, routine account problems, server crashes, etc. For each of these, there is a rate of response that is independent of the nature of the specific request (i.e., which user) and seldom varies from a given average rate μ. Requests that cannot be serviced via an exponential distribution include complex troubleshooting tasks, and any request where the exact method of solution is unknown at the time of request. In general, a request for which the answer is well documented and scripted exhibits an exponential distribution of service times; requests with no documented response do not.

Lessons From Capacity Planning

Real data discussed below shows that inter-arrival times may not exhibit a Poisson distribution, and that service times may not be exponentially distributed. However, much is known about the performance of idealized queues governed by Poisson and exponential distributions, and there are many system administration tasks for which these performance estimates are reasonable.

A queue that exhibits Poisson arrivals with rate λ and has c independent system administrators working with service rates μ is called an "M/M/c" queue. The first M stands for 'memoryless' (Poisson) arrivals, the second M stands for 'memoryless' (exponential) service times, and c is a count of servers (administrators) all of whom complete requests with rate μ. The behavior of an M/M/c queue is well understood and is commonly used for capacity planning of server farms and IT infrastructure.

For an M/M/c queue, whenever $\lambda/c\mu < 1$, the probability that the queue is empty is

$$S_0 = \frac{1}{\sum_{n=0}^{c-1} \frac{(\lambda/\mu)^n}{n!} + \frac{1}{c!}\frac{(\lambda/\mu)^c}{1-\lambda/(c\mu)}} \tag{1}$$

and the "mean time in system" (average wait) for a request [15] is

$$W = \frac{S_0(\lambda/\mu)^c}{c!c\mu(1-\lambda/(c\mu))^2} + \frac{1}{\mu}. \tag{2}$$

The mean time spent waiting for n requests to be serviced is n times the mean wait for one. More important, this equation allows us to predict whether adding more system administrators will not solve a response-time problem. As c grows, the first term of

the above equation goes to 0 and the response time converges toward the theoretical minimum $1/\mu$.

Many other equations and relationships exist for more general queues. In this paper, we will consider only M/M/c models; for an excellent guide to other models and how to predict performance from them (including excel spreadsheets for decision support), see [17].

Learning From Real Data

From above, it is easy to analyze systems that behave according to Poisson arrivals and exponential service. How realistic are these assumptions about system administration? To explore this, we examined a ticket queue from a live site (Tufts ECE/CS). Note that no one knew, until very recently, that anyone was going to analyze this ticket data. It is thus free of many sampling biases. It is, however, difficult to determine exactly when many tickets were closed. This is because there is no site requirement to close tickets promptly, and many tickets are closed by student staff who monitor the ticket queue, sometimes long after the problem has been addressed.

Plotting ticket numbers (an increasing sequence) against time (Figure 3) shows little or no evocative

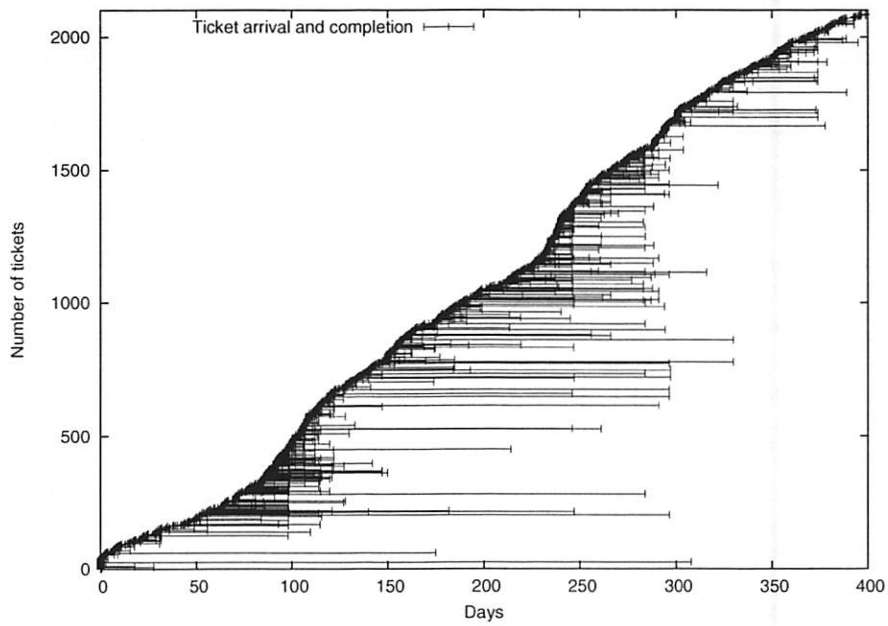


Figure 3: Ticket durations in ECE/CS from 7/2004 to 7/2005.

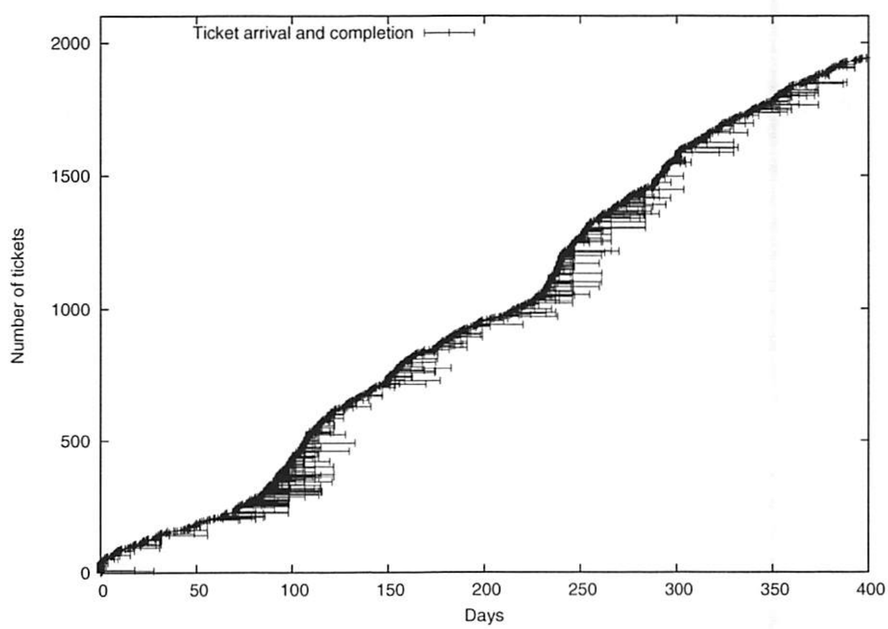


Figure 4: Ticket durations less than 30 days.

patterns. Each ticket is plotted as a horizontal line, with beginning and end representing creation and completion time. The Y axis represents ticket number; tickets due to spam have been deleted and the resulting queue renumbered as increasing integers with no gaps. Note particularly that several tickets are still open after several months.

We discovered very quickly that there were two classes of service: one for short-duration requests and another for long-duration requests. Viewed alone, the requests that took less than a month exhibit relatively consistent response times (Figure 4).

Request arrivals are *not* Poisson. For arrivals to exhibit a Poisson distribution, the mean of inter-arrival times must be equal to their standard deviation. In this case, the overall standard deviation of inter-arrival times (9580 seconds or $\approx$ 2.65 hours) is about 1.37 times the mean (6971 seconds or $\approx$ 1.94 hours), indicating that there are periods of inactivity. Looking deeper, Figure 5 shows one problem: arrival rates are not constant, but instead sinusoidal over a 24-hour period. In this graph, ticket arrivals are shown by hour, summed over the lifetime of the Request Tracker (RT) database. The times are corrected for daylight savings time, and show more intense traffic 9 am to 5 pm with a small dip in traffic at lunch. Ticket closures show a different pattern (Figure 6) with a hotspot at 3 pm that defies explanation, until one realizes that a student administrator charged with monitoring and closing tickets starts work at that time!

Measured "time in system" does not seem to be exponential, either. If, however, one omits requests with time in system greater than one month, the remaining requests exhibit a distribution that looks

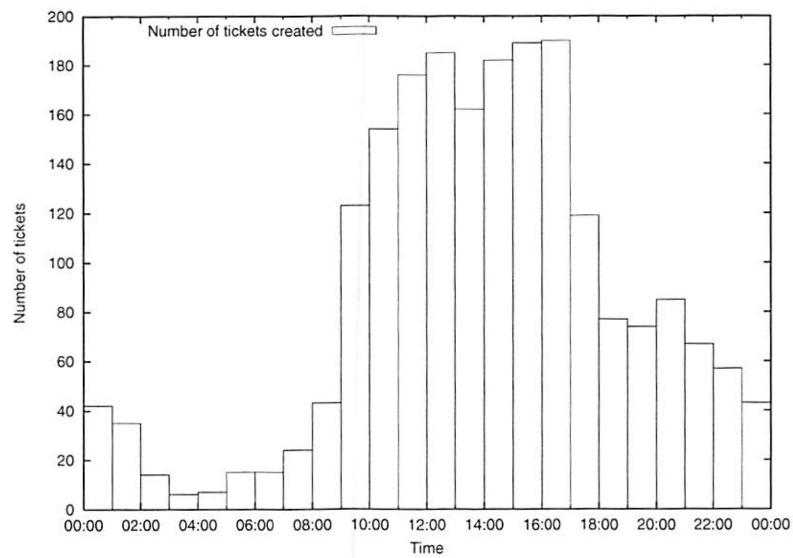


Figure 5: Ticket arrivals exhibit sinusoidal rate variation over 24 hours.

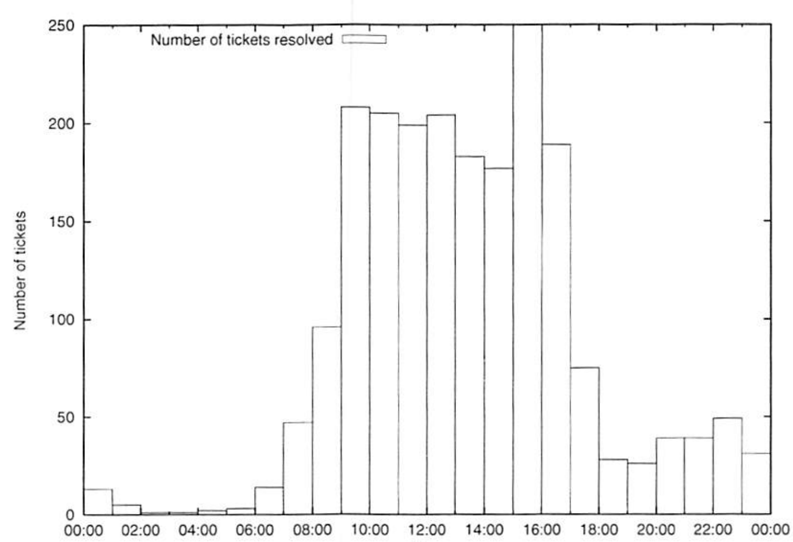


Figure 6: Ticket closures exhibit a sinusoidal time distribution with a hotspot at 15:00-16:00.

similar to exponential (Figure 7). The figure contains a histogram of the number of requests processed in each number of days. Note, however, that this figure represents service time *plus* time spent waiting in queue, so it cannot be used to compute an accurate service rate.

From the data, we see that requests are multiclass with at least two distinct classes of requests:

1. A vast majority of requests are resolved quickly (in less than one month, with a mean time in system of about 3.6 days). Arrival times for these requests seem to be governed by a *sinusoidal non-stationary Poisson process*, i.e., arrival rates seem to vary between a daily high and low on a sine-wave pattern.
2. A small number of requests have an indeterminate and long time in system. Arrival times for these requests show no discernible structure (perhaps due to lack of enough examples).
3. The average rate of ticket arrival is gradually increasing over time. In our case, this seems to be partly due to new faculty hires.

This data also exhibits, however, the main difficulties of extracting performance statistics from ticket queues:

1. Service times are recorded inaccurately because there is no particular advantage to recording them accurately. Most tickets are closed late, because it is not the job of the administrator answering the ticket to close it, but just to solve the problem. In our case, many tickets are closed by student staff some time after the problem has been solved.
2. The class of a particular request is not always easily discernible. It is easier to group requests by time of service rather than class of problem. In our case, there is a clear distinction between requests for which an appropriate response is documented, and those for which an appropriate response is unknown. The former take on average the same time to resolve, while the latter vary widely.
3. Emergent patterns in the data are only obvious if one is very careful about correcting for environmental issues. For example, data did not exhibit a sinusoidal arrival rate until it was corrected for daylight savings time (DST)!
4. Ticket data does not indicate the severity of a problem. There are no discernible "flurries" or "bursts" of data for severe problems; often only one or two people bother to report a major outage.

Other practitioners have mentioned that there are several ways that request queue data can be biased by operating policy.

1. If people are rewarded for closing tickets quickly, they tend to close them early, before an actual resolution.
2. If people are rewarded for only the tickets they choose to resolve, obviously difficult tickets will be avoided.

The final issue plaguing the use of real request queue data is privacy. Real request data contains flaws in practice. For example, some requests for which there should be documented scripts remain undocumented, some requests are forgotten, and some requests can take an embarrassing amount of time to resolve. For this reason, it is difficult for researchers to get real data on the nature of requests and their service times, for sites other than their own.

One lesson learned from our data is the power of good documentation. If an appropriate response to a problem is documented or otherwise well known, there seems to be no significant difference in response time *invariant of the nature of the problem*. It is

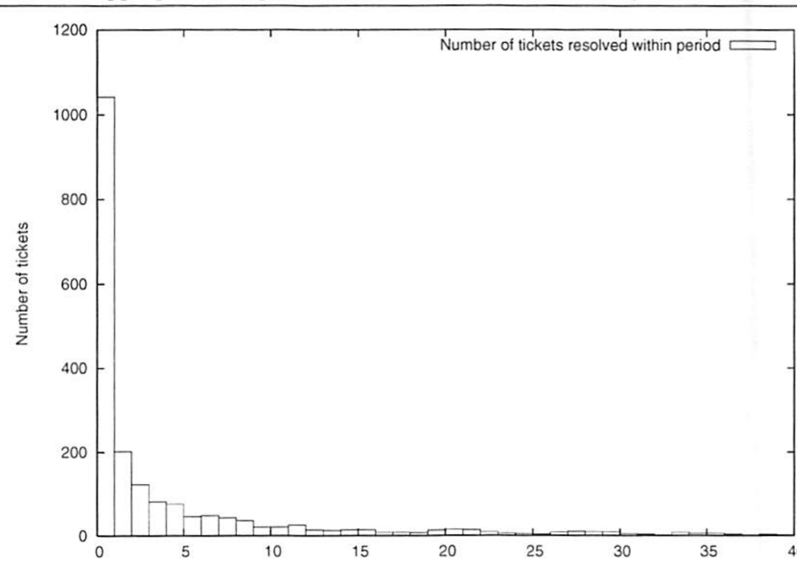


Figure 7: A histogram of the frequency of tickets resolved in each number of days has a near-exponential shape.

surprising that to a first approximation, differences in service times for subclasses of requests do *not* emerge from the data. One possible reason for this effect is that in these cases, communication time with clients may dominate the time needed to solve the problem once it is clearly defined.

Conversely, problems with no documented response wait longer and may never be solved. At our helpdesk, student staff solve routine problems and defer only problems with no documented solution to second-level triage staff. Since the second-level staff are often planning or deploying new architectures, requests without documented solutions await their attention and compete with deployment tasks. Of course, once solved and documented, such a request becomes quickly solvable.

In our data, a simple pattern emerges. System administration is composed of a combination of routine tasks and complex troubleshooting and discovery that borders upon science. Our site's practice is perhaps best described as a two-class queueing system, with a large number of routine requests with documented and/or known solutions, and a smaller number of requests requiring real research and perhaps development. For the most part, the routine requests are accomplished by system administrators acting independently, while more complex requests may require collaboration between system administrators and take a longer, relatively unpredictable time to complete.

A Simple Model Of Cost

Given the above model of system administration as a queueing system, we construct a coarse overall model of cost, based upon the work of Patterson [18] with some clarifications.

First, cost can be expressed as a sum of two components: the "cost of operations" and the "cost of waiting for changes." The "cost of operations" contains all of the typical components of what we normally consider to be cost: salaries, benefits, contracts, and capital expenditures such as equipment acquisition. For most sites, this is a relatively predictable cost that remains constant over relatively long time periods, e.g., a quarter or a year. The "cost of waiting" is a generalization of Patterson's "cost of downtime", that includes the cost of waiting for changes as well as the cost of waiting for outages to be corrected.

While the cost of downtime can be directly calculated in terms of work lost and revenue lost, the cost of waiting for a change cannot be quantified so easily. First we assume that R represents the set of requests to be satisfied. Each request $r \in R$ has a cost C_r and the total cost of waiting is

$$C_w = \sum_{r \in R} C_r \, . \tag{3}$$

We assume that for a request r (corresponding to either an outage or a desired change in operations), there is a cost function $c_r(t)$ that determines the instantaneous cost of not accomplishing the change, and times t_{r1} and t_{r2} at which the change was requested and accomplished. Then the tangible cost of waiting is the integral (running sum) of $c_r(t)$ over the waiting period:

$$C_r = \int_{t_{r1}}^{t_{r2}} c_r(t)dt \, . \tag{4}$$

If as well $c_r(t)$ is a constant

$$C_r = (t_{r2} - t_{r1})c_r = t_r c_r \tag{5}$$

as in Patterson's paper. In general, this may not be true, e.g., if the change reflects a competitive advantage and the effects of competition become more severe over time. For example, in the case of security vulnerabilities, vulnerability is known to increase over time as hackers gain access to exploits.

System administrators control very little of the process that leads to lifecycle cost, but the part they control – how they work and accomplish tasks – can partly determine the cost of waiting. In this paper, we consider the effects of practice upon the cost of waiting in a situation in which the budget of operations is held constant over some period, e.g., a quarter or a year. Situations in which cost of operations can vary (e.g., by hiring, layoffs, or outsourcing) are left for later work.

The cost function $c_r(t)$ must model both tangible (work lost) and intangible (contingency) factors. For requests concerning downtime, the cost of waiting may be directly proportional to work and revenue lost, while for requests involving enhancements rather than downtime, work lost and revenue lost can be more difficult to quantify. Also, the costs of waiting for enhancements vary greatly from site to site. For business sites, delays often incur real revenue loss, while for academic sites, the effects of delays are more intangible, resulting in missed grant deadlines, student attrition, and other "opportunities lost". In the latter case, it is better to model cost as risk of potential loss rather than as tangible loss.

We can best cope with uncertainty and risk by computing the expected value of each potential risk. Contingencies are like requests; they arrive during a period of vulnerability with a specific rate depending upon the severity of the vulnerability; these arrivals are often Poisson. The total expected value of an outage or wait is the sum of expected incident costs, taken over the various kinds of incidents. If incidents arrive with a constant Poisson rate λ, the expected incident cost is the number of expected incidents times the cost of an incident. This is in turn a product of the rate of arrival for the incident, the elapsed time, and the average cost per incident. Note that the word "incident" applies not only to security problems, but also to lost opportunities such as students dropping out of school, employees quitting, etc.

Thus we can think of the cost function $c_r(t)$ for a particular request r as

$$c_r(t) = c_{rm}(t) + c_{ri}(t) \quad (6)$$

where $c_{rm}(t)$ represents tangible losses and $c_{ri}(t)$ represents intangible losses. While $c_{rm}(t)$ represents work and revenue losses and is proportional to the scale of the outage, c_{ri} represents *possible* losses due to random events. If contingencies are elements d of a set D_r of all possible contingencies that can occur during request r, and contingencies in D_r are statistically independent, then the cost c_{ri} for all of them is the sum of their individual costs

$$c_{ri}(t) = \sum_{d \in D_r} c_{rid}(t) \quad (7)$$

where c_{rid} is the contingency cost for $d \in D_r$ while waiting for r. If contingencies $d \in D_r$ have Poisson inter-arrival times λ_d, then

$$c_{rid} = \lambda_d C_d \quad (8)$$

where C_d is the average cost per incident for d. Thus

$$c_r(t) = c_{rm}(t) + \sum_{d \in D_r} \lambda_d C_d. \quad (9)$$

If c_{rm}, λ_d, and C_d are constants, then

$$\alpha_r = c_r m + \left(\sum_{d \in D_r} \lambda_d C_d \right) \quad (10)$$

is also a constant, and

$$C_r = \int_{t_{r1}}^{t_{r2}} c_r(t)dt = \alpha_r t_r \, . \quad (11)$$

Note that there are a lot of if's in the above justification and the reader should be warned that assumptions abound here. The formula for cost of waiting simplifies easily *only if particular assumptions hold.* As we make these assumptions, our model loses accuracy and expressiveness. With all assumptions in place, we have Patterson's model; as he states, it is an oversimplification.

If requests can be divided into classes $k \in K$, each with a different proportionality constant α_k, then the total cost of processing a set of requests is the total time spent waiting for each class, times the proportionality constant for that class. Thus, in the simplest case, the total cost of waiting is

$$C_w = \sum_{k \in K} \sum_{r \in K} \alpha_k t_r \quad (12)$$

or

$$C_w = \sum_{k \in K} \alpha_k \sum_{r \in K} t_r. \quad (13)$$

Thus the contribution of each class k is proportional to the total time spent waiting for events of that class.

In this approximation we make many simplifying assumptions:

1. Contingencies arrive with Poisson rates.
2. Contingencies are statistically independent of one another.
3. The effect of a contingency does not change over time.

These are limits on how we *formulate* a problem; we must not allow dependencies to creep into our classifications. Part of this formulation is to think of bursts of events as single events with longer service times. For example, it is incorrect to characterize "bursty" contingencies such as virus attacks as host events; these events are not independent of one another. However, the event in which the virus entered the system is not bursty, independent of all other like events, and thus can be treated as *one* contingency. Likewise, spam from a particular site is not an independent event for each recipient, though spam from a particular source is often independent of spam from other sources.

The main conclusion that we make from these observations is that

> The intangible cost of waiting for a request is, to a first approximation, proportional to time spent waiting (though the proportionality constant may vary by request or request class).

While some constants remain unknown, the values for some proportionality constants are relatively obvious. If n users are affected by an outage, then the tangible cost of downtime is usually approximately proportional to n. Likewise the rate of incidents that involve one member of the population (such as attrition) is usually approximately proportional to the number of people involved (due to independence of people as free agents).

Estimating Service Rates

In the above sections, we show a linear relationship between the cost of waiting and amount of time spent waiting, and show that the amount of time spent waiting depends upon arrival rate and service rate for tasks. In our observation of real systems, arrival rate was relatively easy to determine. To determine the cost, however, we must also know the service rate with which requests are completed. We cannot measure this parameter directly; we can only measure the waiting time that results from it. How do we estimate the service rate itself? To answer this question, we borrow from a broad body of work on complexity estimation in software engineering [19].

Cost modeling in software engineering concerns the cost of maintaining a large piece of software (such as a program or configuration script). The basic strategy is to measure the complexity of the final product in some way, and then predict from that complexity how much it will cost to craft and maintain the program.

Complexity metrics that can aid in determining cost of a software engineering project include both "white-box" and "black-box" methods. A "black-box" method looks at the complexity of requirements, while a "white-box" method looks at the complexity of a potential finished product. The goal of either kind of analysis is to produce a service rate that can be utilized for later analysis. To map this to system administration, a "white box" method would base cost estimates on the structure of practice, while a "black box" approach would base cost estimates upon the structure of the problem.

White-box Methods

In software engineering, white-box software metrics include:

1. Lines of code (LOC): the complexity of a software product is proportional to its length in lines of code.
2. cyclomatic complexity [16]: the complexity of a piece of software is proportional to the number of "if" statements in the code.

It is generally agreed that cyclomatic complexity is a much better measure of complexity than LOC, for a variety of reasons, including variations in the expressiveness of programming languages; long programs in one language can be short in another. The key is to find something about the program that is more closely related to its cost than its length. For programs, the number of branches contributes to the difficulty of debugging or maintaining the program. The key to white-box analysis of system administration is to find an analogue to the branches for programs.

Whitebox analysis of programs inspires a similar form of analysis for system administration procedures. While white-box analysis of programs starts with pseudo-code, white-box analysis of practice starts with a recipe or instructions to service a particular kind of request. If we treat each recipe as a "program", with "branches" at particular steps, then we can compute the average time taken for the recipe by keeping statistics on the number of times that individual steps are utilized in practice. This provides a way to come up with estimated rates for a procedure, given estimates for subparts of the procedure.

Note that white-box analysis of a recipe for system administration is quite different than white-box analysis of a program. In the case of the program, the white-box measurement of complexity does not depend upon the input to the program. In system administration, the performance of a procedure depends upon the nature of the environment. A white-box estimate of the time needed to service a request is

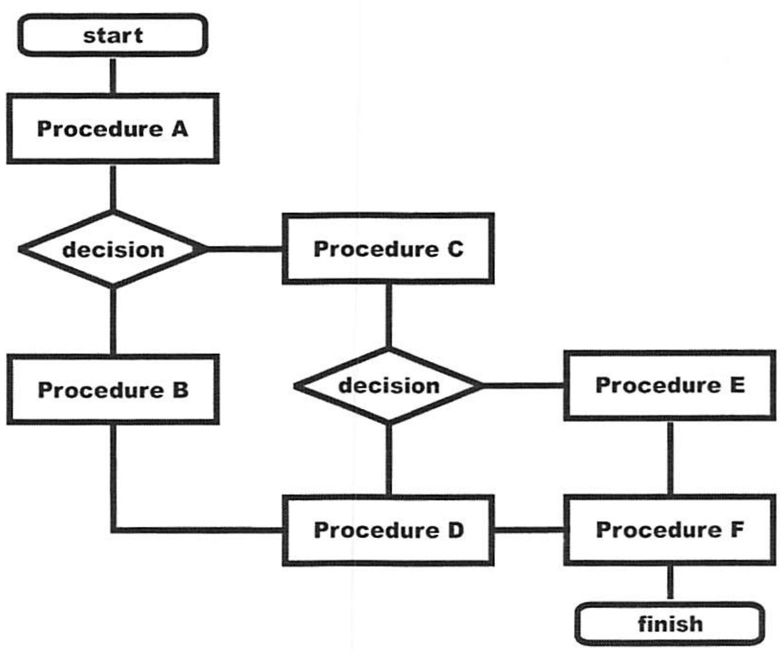


Figure 8: An example troubleshooting flowchart.

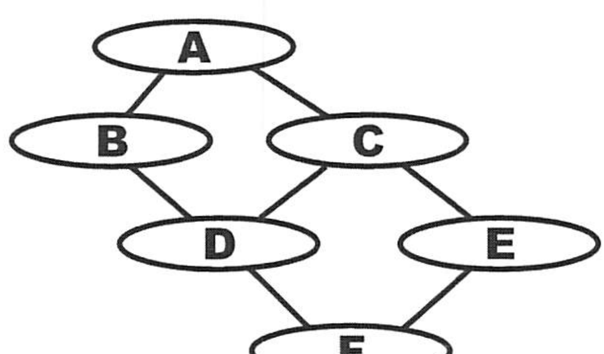


Figure 9: The flow graph corresponding to Figure 8.

a measure of both the complexity of the procedure and the complexity of the environment.

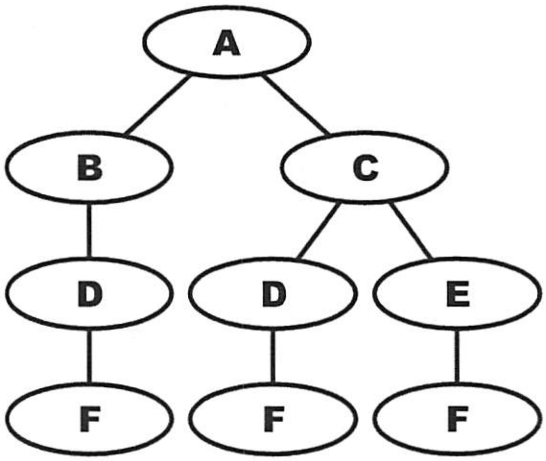


Figure 10: The flow tree corresponding to Figure 9.

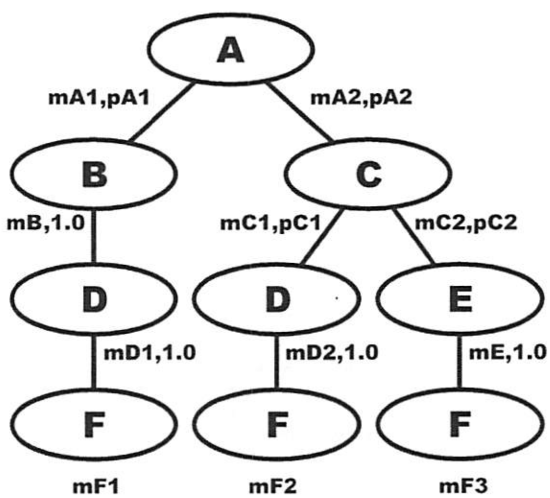


Figure 11: An annotated flow tree tracks statistics that can be used to compute average completion rate.

One way of performing white-box analysis starts with an (acyclic) troubleshooting chart for a procedure. We start with a a troubleshooting chart (Figure 8) that describes procedures to perform and branches to take after each procedure. We convert this to a flow graph (Figure 9) by representing only decision nodes. Since a typical troubleshooting chart has no loops, we convert this graph into a flow tree by duplicating nodes with two or more incoming edges (Figure 10). We then annotate branches in that tree with statistics to be collected or estimated about the branch (Figure 11). These statistics allow us to compute the mean service rate for the whole tree.

The key to the computation is that given that we know a service rate for the subtrees of a node in the tree, we can compute the service rate for the node itself. The nature of the computation is illustrated in Figure 12. Suppose we know the service rate mB for subtree B and mC for subtree C. Suppose that we want to compute the service rate mA for A, and know for each branch out of A, the service rate for A given that it takes the branch (mA1,mA2) and the probability with which that branch is taken (pA1,pA2). If we take the branch from A to B, and A has service rate mA1, then the average service time for the branch is 1/mA1 + 1/mB. If we take the branch from A to C, the average service time for the branch is 1/mA2 + 1/mC. If we take the branch to B with probability pA1, and the branch to C with probability pA2, then the average service time for both is the expected value

$$\mathrm{pA1}\left(\frac{1}{\mathrm{mA1}}+\frac{1}{\mathrm{mB}}\right)+\mathrm{pA2}\left(\frac{1}{\mathrm{mA2}}+\frac{1}{\mathrm{mC}}\right). \qquad (14)$$

Thus the average *rate* is the reciprocal of this.

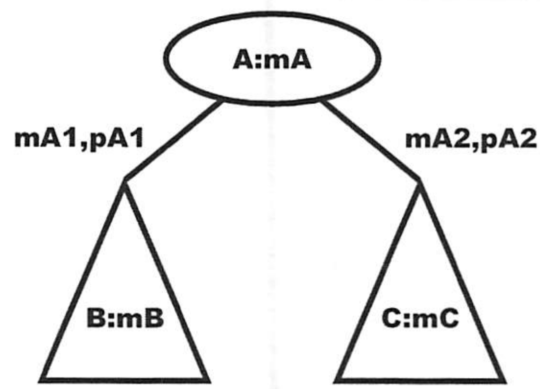


Figure 12: Computing average completion rate for a flow tree.

To enable this computation, each edge in the program graph is labeled with two quantities: the mean service rate for the predecessor of the edge, given that this branch is chosen, as well as the probability that this branch is chosen. We can either measure these directly or estimate them by some method. One obvious way to measure both rates and probabilities is to perform the procedure many times and record the number of times each node is visited, the average service time before taking each branch, and the number of times each branch is taken. Then the ratio of the times the branch is taken, divided by the times its parent is visited, is the sample probability that the branch will be taken.

In this abstraction there hides an astonishing fact: the order in which decisions are made strongly affects the time-in-system for such a graph. While the rates are properties of the administrative *process*, the probabilities of branching are properties of the *environment*. Further, these probabilities are not conditional in the Bayesian sense; they are *temporo-conditional* in that they depend upon the previous occurrence of a specific procedure. In Figure 11, the probability of going to B from A is not the conditional probability $P(B|A)$, but the probability of *B after A*: the probability that we

choose B given that A has already been completed. Bayesian identities do not hold; any change in a procedure affects the sample probabilities of all branches *after* it in the script.

One way to estimate branch probabilities in this model is that certain subtasks depend upon heterogeneity that is routinely tracked. For example, one step might be to determine whether the affected host runs Solaris or Linux. In this case, the sample probabilities for the branches are both known in advance from inventory data. In the same way, one can estimate some branch probabilities from overall statistics on the sources of trouble within the network.

Black-box Methods

White-box methods depend upon the fact that the nature of practice is already known, i.e., we know the steps that people will take to accomplish tasks. In system administration, as in software, we are often asked to estimate the cost of a process without knowing the steps in advance. To do this, we must use a method that estimates cost based upon the complexity of the outcome rather than the complexity of the process.

Black-box methods for measuring software complexity include COCOMO [2, 3, 4]: the complexity of software depends upon an overall characterization of the software's character and mission. COCOMO depends upon use of one of two estimations of code complexity:

1. "object points" [3, 4]: the complexity of a piece of software is proportional to the complexity of the objects it must manipulate.
2. "function points": the complexity of a piece of software is proportional to the number of functions that it must perform.

The key idea in COCOMO is that there is a relationship between the cost of maintaining a program and the complexity of its interactions with the outside world, *though we may not know the exact nature of that relationship in advance.* COCOMO is "tuned" for a specific site by correlating object or function points with actual costs of prior projects. COCOMO is site-specific; the relationship between complexity and cost varies from site to site. By computing a ratio estimating the relationship between requirements and capabilities, one estimates the time that will be taken to complete requirements.

We can apply the idea of COCOMO to system administration in a very direct way. While the software model for function points considers open files, the analogous concept of function points for a network service would describe that service's dependencies and interrelationships with others. We know that the number of dependencies within a service influences the cost of maintaining it; we do not know the exact relationship.

For example, we might compute the function points for an apache web server by assigning a number of points to its relationship with each of the following subsystems: the network (DHCP, DNS, Routing), the filesystem (protections, mounts), and the operating system (users and groups). In a function point model, each of these attributes is assigned a "weight" estimating how difficult it is for a system administrator to deal with that aspect of configuration and management. The sum of the weights is then an estimator of "how complex" the service will be to manage.

The main difficulty with this approach is the number of potential weights one must estimate; virtually every subsystem mentioned in the system administrator's book of knowledge [11, 13] has to be assigned a weight. Further, these weights are not universal; they vary from site to site, though it is possible that similar sites can use similar weights. For example, weights assigned to subsystems vary greatly with the level of automation with which the subsystem is managed.

The cost of providing any service depends not only upon the complexity of the service, but also upon the capabilities of the staff. Our next step in defining a function point estimate of the complexity of system administration is to derive a capability summary of the administrative staff and site in terms of service rate. Obviously, a more experienced staff deals with tasks more effectively than a less experienced one. Capabilities in the model might include end-user support, service support, architecture, etc. If each staff member is assessed and the appropriate attributes checked, and a sum is made of the results, one has a (rough) estimate of capabilities of one's staff. This has similarities to the SAGE levels of system administrator expertise defined in the SAGE booklet on job descriptions [9].

The last step in defining a function point estimate of the complexity of system administration is to assess the capabilities maturity of the site itself. One might categorize the site into one of the following maturity categories [14]:

1. ad-hoc: everything is done by hand.
2. documented: everything is documented, no automation.
3. automated: one can rebuild clients instantly.
4. federated: optimal use is made of network services.

Again, each one of these has a weight in determining the overall capabilities. The sum of administrator capabilities and site capabilities is an estimate of overall "capability points" for the site.

It can then be argued that the complexity of administering a specific subsystem can be estimated by a fraction

$$\text{service rate} = \frac{\text{estimated service points}}{\text{estimated capability points}} \tag{15}$$

where service points and capability points are sums of weighted data as described above. If the weights for capability points are rates in terms of (e.g.) service-points per hour, then the complexity is the average response time in hours to a request.

The overwhelming problem in tuning COCOMO for system administration is that tuning the model requires detailed data on actual measured rates. The tuning process requires regression to determine weights for each of the complexity and quality factors. This is accomplished by studying a training set of projects with known outcomes and properties. To use COCOMO-like systems, we must be able to gather more accurate data on the relative weights of subsystems than is available at present.

Some Experiments

So far, we have seen that we can estimate the cost of system administration via one of two models. "Black box" methods require that we assess the time impact of the complexities of the problem being solved, while "white box" methods require that we estimate the time taken for a detailed set of tasks. Of these methods, "black box" methods give us information more quickly, but these methods require that we "score" facets of the problem being solved as harder or easier than others. These scores must be developed via practice, but we can learn something about the relative complexity of black-box attributes via simulation. By simulating practice, we can account for realistic human behaviors that cannot be analyzed via known queueing theory. We can also observe how real systems can potentially react to changes in a less ideal way than ideal queueing models suggest. Particularly, we can study behavior of queueing systems "on the edge"; almost out of control but still achieving a steady state. In our view, this situation describes more IT organizations than it should.

The Simulator

The simulator, written in C++, is basically an M/M/c queue simulator with the ability to simulate non-ideal ("non-product") behaviors. It assumes that we have c identical system administrators working 24x7 and generates a variety of requests to which these ideal administrators must respond. One can vary the behavior of the system administrator and the request queue and measure the results. The input to the simulator is a set of classes of tasks, along with arrival and service rates for each task. The output is the time spent waiting for requests (by all users), both per time-unit of the simulation and overall. We assume for this simulator that the cost of waiting is a constant; a unit wait by a single individual results in some constant intangible cost. These simulator assumptions are very much less complex than real practice, but one can make some interesting conclusions from even so simple a model.

Diminishing Returns

Our first simulation exercise is to study the effects of adding system administrators to a group servicing simple requests. We set up a simple multi-class system with a varying number of system administrators all of whom have identical average response rates. There are four request classes, corresponding to requests whose service time is an average of 1, 3, 8, and 24 hours, respectively. The service rate of each request class is twice its arrival rate, creating a balance between arrivals and service. We ran the exact same simulation for two, three, and four system administrators. The cumulative time spent waiting for service is shown in Figure 13. There is clearly a law of diminishing returns; the change in wait time from three to four system administrators does not significantly change the time spent waiting for service.

Saturation

Realistic system administration organizations can be faced with request volume that is impossible to

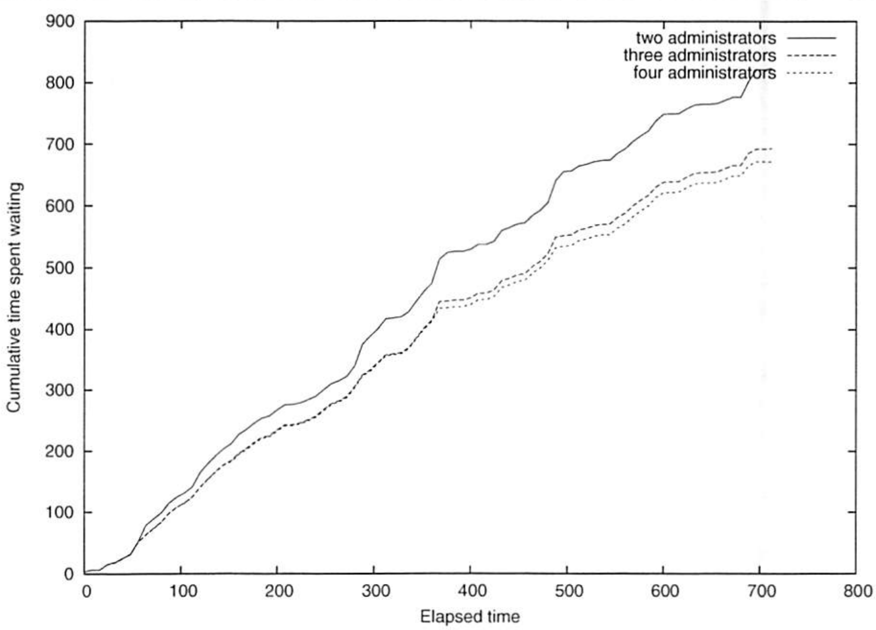


Figure 13: Diminishing returns when adding administrators to a queue.

resolve in the time available. We know from classical queueing theory that an M/M/c queuing system exhibits predictable response time only if $\lambda/c\mu < 1$, where λ is the arrival rate, c is the number of administrators, and μ is the service rate per administrator. In other words, there has to be enough labor to go around; otherwise tickets come in faster than they can be resolved, the queue grows, and delays become longer and longer as time passes.

Figure 14 shows the same simulation as before, but adds the case of one administrator. This seems like an unbalanced situation in which request rate is greater than service rate, but looking at waiting time per unit time (Figure 15) we see that waiting time is not always increasing with time. So although one administrator is very much slower than two or three, the situation is not completely out of control. Note, however, that the situation of the single administrator is very sensitive to load; he is "on the brink of destruction." Small changes in load can cause large variations in response time, and the cost of administration due to waiting is "chaotic", especially when a request when a long service time enters the queue. Nevertheless, on average, the time spent waiting varies directly with elapsed time of the simulation.

Figure 16 shows incremental waiting time for a truly "saturated" system in which there is no way for administrators to ever catch up. We saturate the queue in the previous example by multiplying the arrival rates for all requests by four. In this case, one and two administrators are in trouble; queue length is growing linearly with time along with wait time. Figure 17 shows the cumulative time for this example. In a saturated queueing system, since time spent waiting

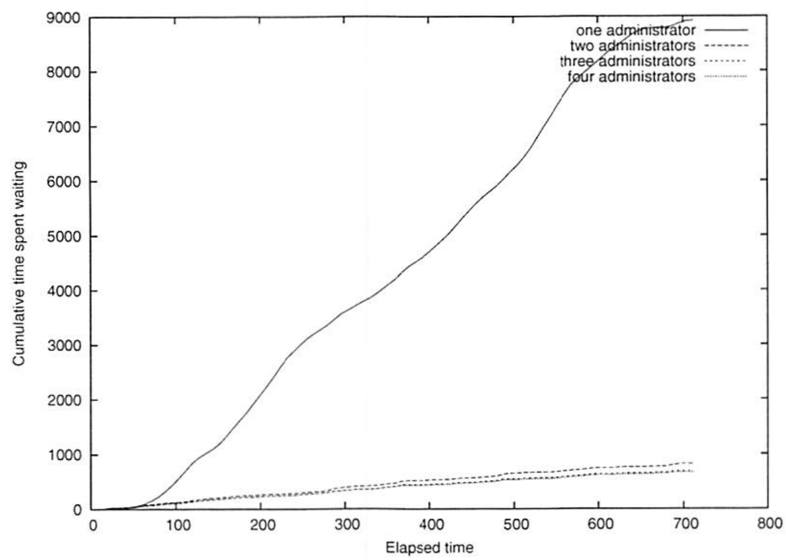


Figure 14: One administrator performs very poorly compared to two, three, and four.

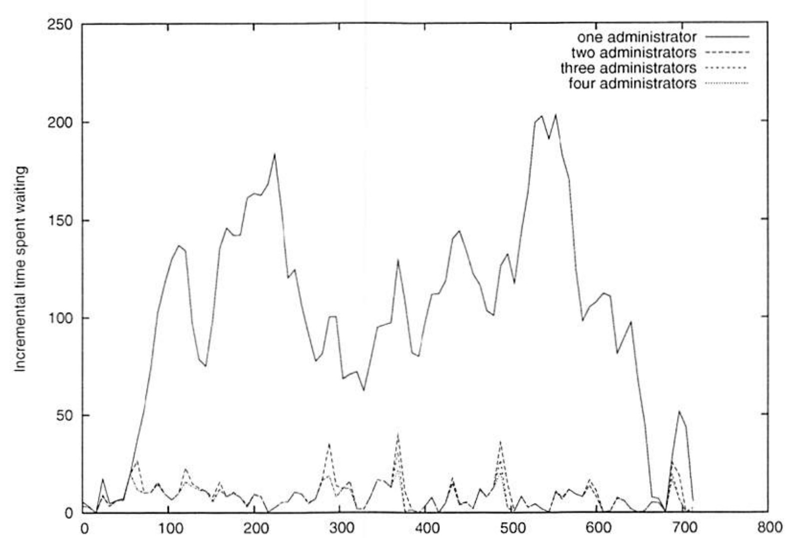


Figure 15: Incremental data for Figure 14 shows that utilizing one administrator leads to chaotic wait times.

increases linearly with elapsed time, the cumulative time spent waiting varies as the *square* of elapsed time.

Brinksmanship

We consider it a fair statement that many IT organizations run with $\lambda/c\mu$ quite close to 1. It is thus no surprise that it is very difficult for these organizations to cope with changes that might increase load upon system administrators, even for a short time. There is a solution, but it is counter-intuitive. Figure 18 shows the effect of a "catastrophic" flurry of requests arriving in a near-saturated system. For a short while, wait times go way up, because the system is already almost saturated and the new requests push it over the limit. The key is to distribute the same requests over a long time period (Figure 19), to avoid pushing the system over the limit and save waiting time. Note that in both figures, one administrator alone simply cannot handle the load and chaotic waits occur in both cases.

Lessons Learned

Human systems self-organize around maximum efficiency for the task at hand, but not necessarily for future tasks. As such, we as system administrators are often near the "saturation point" in our practice. As in Figure 18, small increases in load can lead to catastrophic delays. But the strategies in Figure 19 can help.

One part of coping with being understaffed is to utilize automation to lessen workload, but that can lead to queue saturation in an unexpected way. The quandary of automation is that when something goes wrong, it is not one host that is affected, but potentially hundreds. If an automation mistake affects hundreds of nodes, we often have the situation in Figure 18; there are hundreds

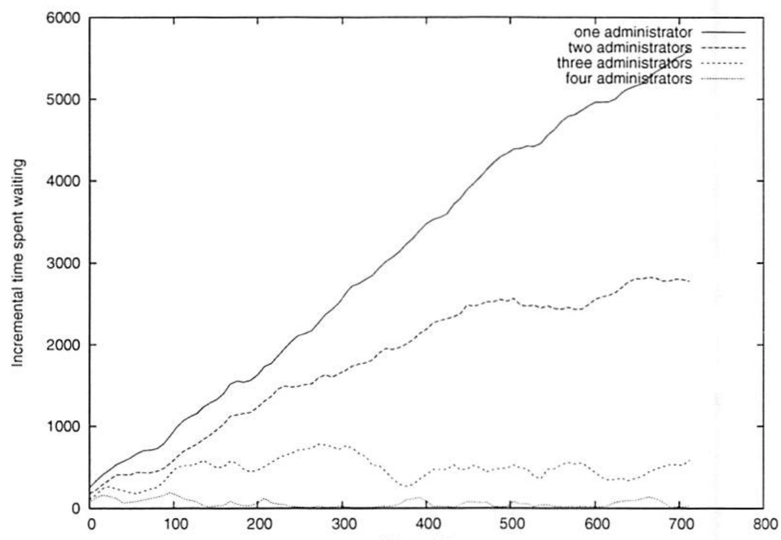


Figure 16: Multiplying the arrival rate by four overloads one or two system administrators.

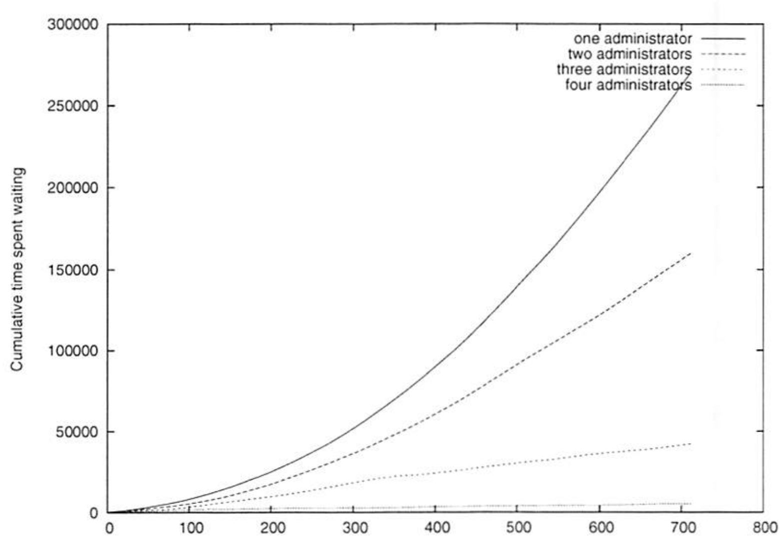


Figure 17: Cumulative wait time for overloaded system administrators varies as the square of elapsed time.

Voluntary Cooperation in Pervasive Computing Services

Mark Burgess and Kyrre Begnum – Oslo University College

ABSTRACT

The advent of pervasive computing is moving us towards a new paradigm for computing in terms of *ad hoc* services. This carries with it a certain risk, from a security and management viewpoint. Users become increasingly responsible for their own hosts. A form of service transaction based on minimal trust is discussed. A proof of concept implementation of non-demand (voluntary) services is discussed for pervasive computing environments. 'Voluntary Remote Procedure Call' is a test-implementation of the proposed protocol integrated into cfengine, to show how voluntary cooperation of nodes can allow a cautious exchange of collaborative services, based on minimal trust. An analysis of implementation approaches followed by a discussion of the desirability of this technology is presented.

Introduction

Pervasive or ubiquitous computing is often wedded to the future vision of mobile and embedded devices, in smart homes and workplaces; however, pervasive computing environments already exist today in Web Hotels and at Internet Service Providers. A changing base of customers meets in an environment of close proximity and precarious trust and offers services.

Pervasive mobile computing presents two independent challenges to content services. The first concerns how to secure platforms for virtual commerce. Here the problem is that one does not possess guaranteeable credentials for those connecting to the services. The second concerns how to deal safely with anonymous, non-commercial services offered entirely on an ad hoc, cooperative basis. Here the problem is the risk involved in interacting at all with a client of unknown intentions.

The former generally drives discussions about security and public key infrastructures (though this is of little help unless every user is independently verified and identifies cannot be forged). The latter has made a name for itself through peer to peer file sharing such as Napster, Gnutella and other services. There is currently no technological solution that can determine the intentions of a client attempting to connect to a service.

The challenges of pervasion are not only technical, but also 'political'. *Autonomy* is a noose by which to hang everything from security to system management in coming years. In the foreseeable future, one can expect pervasive services to be vying for attention on every street corner and in every building. The key ingredient that makes such a scenario difficult is *trust*. Orderly and predictable behaviour is a precondition for technology that performs a service function, but if one connects together humans or devices freely in environments and communities that share common resources, there will always be conflicting interests. Anarchy of individual interests leads to contention and conflict amongst the members of the community; thus, as in the real world, an orderly policy of *voluntary cooperation* with norms is required that leads to sufficient concensus of cooperative behaviour.

In this paper, we shall be especially interested in the administrative challenges of pervasion, such as services like backup, software updates, policy updates, directory services etc. We shall attempt to tackle a small but nevertheless important problem, namely how to 'approach' peers for whom one does not have automatic trust.

In a pervasive setting, the the sheer multiplicity of devices and hosts demands autonomous administration, for scalability. This means that decisions about trust cannot be reasonably processed by a human. Self-managing 'smart' devices will have to cope with decision-making challenges normally processed by humans, about the trustworthiness and reliability of unknown clients and servers. Devices must make political decisions, based on often fuzzy guidelines. This is not a problem that can be solved by public key infrastructures, since apparent certainty of identity is no guarantee for trust.

Virtualization and the Multiplicity of Autonomous Agents

In a few years, most services could well be provided by virtual machines running on consolidated computing power [1, 2], leading to an even denser packing of competing clients and services. There are distinct advantages to such a scenario, both in terms of resource sharing and management. Users can then create and destroy virtual machines at will, in order to provide or consume services or manage their activities.

The ability to spawn virtual clients, services and identities, quite autonomously, makes the interactions between entities in a pervasive environment full of

uncertainties. Rather than having a more or less fixed and predictable interface to the outside world, as we are used to today, users will become chameleons, changing to adapt to, or to defy, their neighbours' wishes. What users choose to do with this freedom depends more on the attitudes of the individuals in the virtual society than on the technologies they use.

Today, at the level of desktop computing, the trend from manufacturers is to give each user increasing autonomy of administrative control over their own workstations, with the option to install and configure software at whim. Thus each user can be either friend or foe to the community of behaviours – and the uniformity of policy and configurations that many organizations strive for today will not be a certain scenario of the future.

We are thus descending into an age of increasing autonomy and therefore increasing uncertainty about who stands for what in the virtual society. The aim of this paper is to offer a simple mechanism for host cooperation, without opening the hosts to potential abuses. It does this by allowing each individual agent to retain full control of its own resources.

The plan for the paper is as follows: we begin with a discussion of the need for cautious service provision in a pervasive environment, by using what is known about client-server relationships between humans. Some examples of management services are discussed that present a risk to both parties and it is suggested how a consentual, voluntary protocol might be implemented to reduce the risk to an acceptable level. A solution to negotiating voluntary cooperation is presented, which trades the traditional model of vulnerable guaranteed service levels for increased security at the expense of less predictable service rates. An implementation of the protocol in the programming language Maude helps to enlighten the possible administrative obstacles which follows of the protocol. An example implementation in the system administration agent framework cfengine is presented which incorporates voluntarism in the existing cfengine communication framework. A comparison and discussion of the two approaches concludes the paper.

Client-server Model

In the traditional model of network services, a client stub contacts a server on a possibly remote machine and requests a response. Server access controls determine whether the request will be honoured and server host load increases as a function of incoming requests. Clients wait synchronously for a reply. This is sometimes called a Remote Procedure Call (RPC).

This model is ideal for efficiently expediting services that are low risk and take only a short time to complete, but it has a number of weaknesses for services that require significant processing or have basic administrative consequences for the network: there is thus the risk of exploitation and denial of service vulnerabilities.

Management services are not typical of network services in general. By definition they are not supposed to be high volume, time critical operations such as web services or transaction processing services. They are nonetheless often of great importance to system security. In a pervasive computing environment, hosts are of many different types; many are mobile, partially connected systems under individual control, configured at the whim of their owner. A more defensive posture is therefore called for in the face of such uncertainty.

This paper presents a simple protocol that assumes no automatic trust: a remote service model based entirely on 'pull' semantics, rather than 'push' semantics. Thus services can be carried out, entirely at the convenience of the answering parties: the risk, at each stage, is shifted to the party that is requesting communication. One interesting side-effect of this is that services become more resilient to Denial of Service attacks.

Pervasion and Stable Policies

The interactions between hosts can naturally lead to changes in their behaviour, e.g., if they exchange configurations, policy instructions or even software. In the ad hoc encounters that emerge between politically autonomous hosts, some stability must condense out of the disorder if one is to maintain something analogous to law and order in the virtual realm [3]. Law and order can only be built on cooperative concensus – it can only be enforced if a majority wants it to be enforced, and conforms in a scheme to enforce it. How such a concensus emerges is beyond the scope of the present paper, but will be discussed elsewhere [4, 5, 6].

Tourism

In the present scenario, the model for contact between computers is like virtual tourism: visitors roam through an environment of changing rules and policies as guests, some of them offering and requesting services, like street sellers or travelling salesmen (no relation to the famed algorithmical optimizer). The communication in such encounters is de-centralized; it takes the form of a "peer-to-peer" relationship between participants. The snag with peer-to-peer transactions that it requires considerable trust between the participating hosts. Who is to declare that an arbitrary participant in the scheme is trustworthy [7]?

It is known from game theoretical and evolutionary simulations that stability in groups of collaborative agents is a precarious phenomenon, and thus caution in accepting arbitrary policy instructions is called for. If hosts alter their behaviour automatically in response to ad hoc interactions, then the effective behaviour of the group can easily be subverted by a rogue policy [8] and the illusion of organizational control can evaporate.

The approach to collaboration which is adopted below is to assume the scenario of lawlessness mentioned above, by placing the provision of administrative

services (e.g., backups, information or directory services, policy distribution etc.) on an entirely voluntary basis. This is, in a tangential way, a continuation of the idea of the "pull contra push" debate in system administration [9, 10] (see Figure 1).

Voluntary Cooperation

There are three reasons why the traditional model of client-server communication is risky for pervasive computing services (see Figure 1).

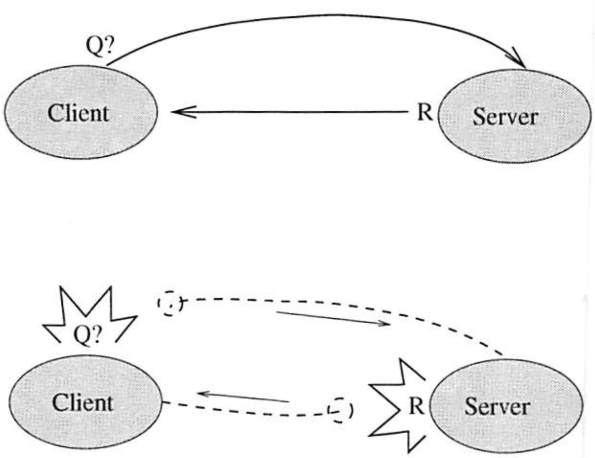


Figure 1: In a traditional client-server model, the client drives the communication transaction "pushing its request". In voluntary RPC, each party controls its own resources, using only 'pull' methods.

1. Contemporary network servers are assumed to be 'available'. Clients and servers do not expect to wait significantly for processing of their requests. This *impatient expectation* leads to pressure on the provider to live up to the expectations of its clients. Such expectation is often formalized for long term relationships as Service Level Agreements (SLA). For ad hoc encounters, this can be bad both for the provider and for the client. Both parties wait synchronously for a service dialogue to complete, during which their resources can easily be tied up and exploited maliciously (so-called Denial of Service attacks).

   Non-malicious, random processes can also be risky. It is known from studies of Internet traffic that random arrival processes are often long tailed distributions, i.e., include huge fluctuations that can be problematical for a server [11]. Providers must therefore over-dimension service capacity to cope with seldom events, or accept being choked part of the time. The alternative is to allow asynchronous processing, by a schedule that best serves the individuals in the transaction.
2. The traditional service provider does not have any control over the demand on its processing resources. The client-server architecture drives the servers resources at the behest of the client. This opens us to risk of direct attacks like Denial of Service attacks and load storms from multiple hosts. In other words, ad hoc encounters do not have to trust only individuals but also the entire swarm of clients in a milieu collectively. The alternative is that hosts should be allowed to set their own limits.
3. The traditional service provider receives requests that are accepted trustingly from the network. These might contain attacks such as buffer overflows, or data content attacks like viruses. The alternative is for the server to agree only to check what the client is offering and download it if and when it can be handled safely.

Although one often aims to provide on-demand, high-volume services where possible, it is probably not prudent to offer any service to just any client on demand, given how little one actually can know about them. Nor is it necessary for many tasks of a general administrative nature, where time is not of the essence, and a reply within minutes rather than seconds will do (e.g., backups, policy updates, software updates etc.). Avoiding messages from a client is a useful precaution: some clients might be infected with worms or have parasitic behaviour.

The dilemma for a server is clearly that it must expose itself to risk in order to provide a service to arbitrary, short-term customers. In a long term relationship, mutual trust is built on the threat of future reprisals, but in an opportunistic or transitory environment, no such bond exists. Clients could expect to 'get away' with abusing a service knowing that they will be leaving the virtual country soon.

In the voluntary cooperation model this risk is significantly reduced in the opening phase of establishing relations: neither "client" nor "server" demand any resources of each other, nor can they force information on each other. All cooperation is entirely at the option of the collaborator, rather than at the behest of the client. Let us provide some examples of how voluntary collaboration might be used,

Configuration Management in a Pervasive Setting

A natural application for voluntary cooperation is the configuration or policy management of networked hosts. How shall roaming devices adapt to what is local policy [12]? A host that roves from one policy region to another might have to adapt its behaviour to gain acceptance in a scheme of local rules, just as a traveller must adapt to the customs of a local country. However, this does not imply automatic trust (see Figure 2). Today, many users think in terms of creating a Virtual Private Network (VPN) connection back to their home

base, but to get so far, they must first be allowed to interact with their local environment.

Configuration management is a large topic that concerns the configuration of resources on networked hosts. See, for instance, [14] for a review. There are several studies examining how computing systems can be managed using mobile agents. However, mobile agents have not won wide acceptance amongst developers [15] and they violate the risk criteria of the present paper. Opening one's system to a mobile agent is a potentially dangerous matter, unless the agent follows a methodology like the one described here.

Three examples of autonomy in computer administration are shown in Figure 2. In (a) all instructions for configuration and behaviour originate from a central authority (a controller). This is like the view of management in the SNMP model [16, 17] and traditional telecom management models like TMN [18]. The dotted lines indicate that nodes in the network could transmit policy to one another to mitigate the central bottleneck; however, control rests with the central controller, in the final instance. In (b) there is a hierarchical arrangement of control [19, 20]. A central controller controls some clients individually, and offers a number of lesser authorities its advice, but these have the power to override the central controller. This allows the delegation of control to regional authorities; it amounts to a devolution of power. In (c) one has a completely decentralized (peer to peer) arrangement, in which there is no centre [21, 3]. Nodes can form cooperative coalitions with other nodes if they wish it, but no one is under any compulsion to accept the advice of any other.

This sequence of pictures roughly traces the evolution of ideas and technologies for management. As technology for communication improves, management paradigms become naturally more decentralized. This is probably more than mere coincidence: humans fall naturally into groups of predictable sizes, according to anthropologists [22]. There is a natural rebellion against centralization once the necessary communication freedoms are in place. Then groupings are essentially "buddy lists", not central authorities.

A decentralized scheme has advantages and disadvantages. It allows greater freedom, but less predictability of environment. Also, by not subscribing to a collaborative authority, one becomes responsible for one's own safety and well-being. This could be more responsibility than some "libertarian" nodes bargain on. One could end up re-inventing well-known social structures in human-computer management.

The immunity model of system administration describes how nodes can made to take responsibility for their own state, in order to avoid flawed external policies, bottle-necked resources and unnecessary assumptions about trust [10]. The trouble with making hosts completely independent is that they need to communicate and even 'out-source' tasks to one another. The immunity model must therefore extend to communications between hosts that allow them to maintain their autonomy, accepting and rejecting data as they please. Other suggestions for an RPC paradigm in a mobile environment include refs. [23, 24, 25].

Some examples applications for voluntary Remote Procedure Call (RPC) are presented below. Most of these address the impossibility of establishing true nature of a client in an ad hoc meeting.

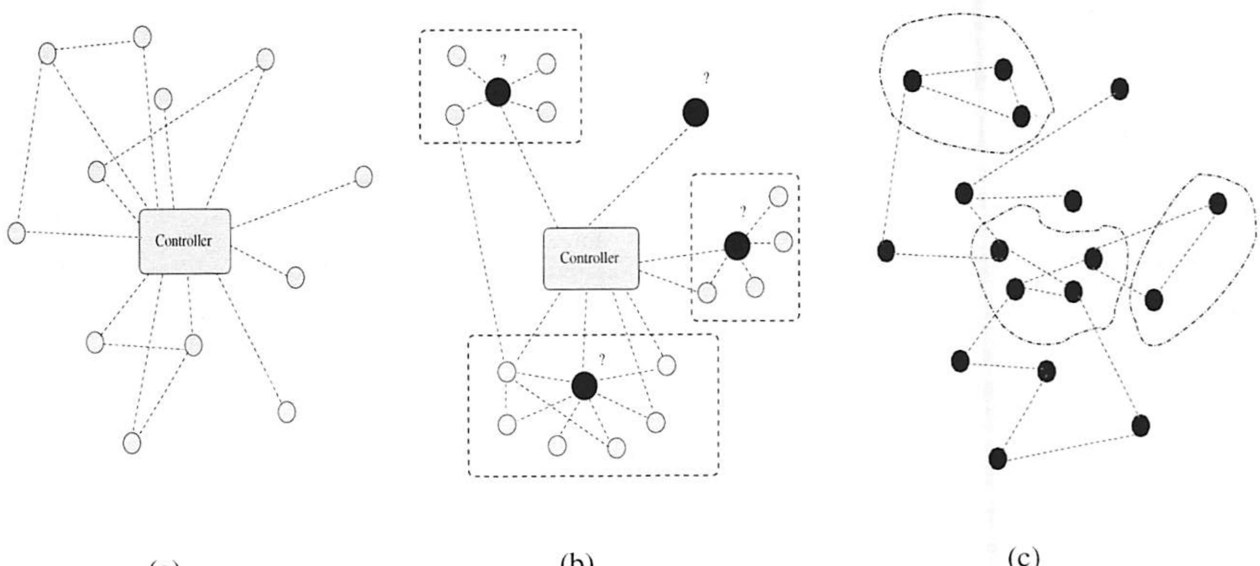


Figure 2: Three forms of management between a number of networked host nodes. The lines show a logical network of communication between the nodes. Dark nodes can determine their own behaviour, i.e., they have political freedom, while light nodes are controlled. The models display a progression from centralized control to complete de-centralization of policy (see [13, 3]).

1. Mobile hosts can use voluntary collaboration to check in to home base for services. For example, a laptop computer that requires a regular backup of its disk to home base, using a pull-only mechanism will need to check in to home base because the server does not know its new IP address, or whether it is even up and running. (Using IPv6, a host could register its mobile address with home base to set up IPv6 forwarding, but if a device has no home base, then this will not work.) The laptop would like to control when its resources are going to be used by the backup software and vice versa, rather than being suddenly subjected to a highly resource intensive procedure at an arbitrary time.

   With a voluntary agreement, the resulting backup time is a compromise between the clients wishes and the server's wishes. It is not exactly predictable in time for either party, but it is predictable in its security. Hosts must learn the expected services times rather than assume them.

   (Again, it is emphasized that encryption cannot solve this problem. Merely opening an encrypted tunnel back to home base does not help the parties to manage their resources. It only helps them to establish their respective identities and to communicate privately.)
2. Exchanges of security alerts and patches could be offered on a voluntary basis, e.g., warnings of recent port-scanning activity, detected anomalies or denial of service (including spam) attacks. Such warning could include IP addresses so that hosts could add the offending IP-port combinations to its firewall. Using this scheme, client hosts agree to signal one another about changes and patches which they have received. These could be cross referenced against trusted signature sources downloaded separately – allowing an impromptu Trusted Third Party collaboration. This kind of technique is now being used for online operating system updates.
3. The ultimate autonomy in distributed services is the free exchange of information and services between arbitrary hosts [12]. Voluntary acceptance of policy or method data could be utilized as a 'cautious flooding' mechanism, for 'viral spreading' of data; e.g., in which each host signals its neighbours about current policy or software updates it has received, allowing them a controlled window of opportunity to download the software (rather than simply opening its ports to all hosts at the same time). It is known that such mechanisms are often efficient at spreading data [26, 27]
4. Neighbouring Service Providers with a standing arrangement with one another could use the voluntary mechanism to sell each other server capacity, using virtual machine technologies. When the demand on ISP1 becomes too high, a method SpawnVM() which is sent to the collaborator ISP2. If ISP2 has the capacity to help, it spawns a virtual machine and returns the IP address. The method call could also include a specification of the system.

These examples underline the need for individuals to maintain control of the their own resources, but at the same time allow one another the potential to collaborate.

Protocol

There are two phases of voluntary cooperation: agreeing to work together (forming a contract) and fulfilling one's obligations (if any) with minimal risk to oneself. Then, given an autonomous framework the implementation of a cautious service protocol is straightforward.

The Negotiation Phase

A willingness to cooperate between individuals must be established in any service before service provision can be considered. This is one of the crucial phases and it brings us back to the opening remarks. Apart from the traditional client request, there are two ways to set up such an agreement: with or without money. One has the following options:

1. A contract based on money: a client is willing to pay for a service and some form of negotiation and payment are processed, based on a legal framework of rules and regulations. This is a common model because it is rooted in existing law and can threaten reprisals against clients who abuse privileges by demanding a knowledge of their identity in the real world.
2. No contract but registration: clients use the service freely provided they register. Again, they must surrender some personal details about themselves. This is a declaration of trust. A password or secret key can be used to cache the results once a real world identity has been established. Public key technologies are well acquainted with the problem of linking true identity to a secret key. We shall not discuss this further but assume the problem to be soluble.
3. Open connection: the traditional way of connecting to host services requires no registration or credentials other than an IP address that may or may not be used to limit access.

In volume services, one usually wants to know in advance which clients to expect and what their demands on service are: this expectation is guaranteed by access control rules, firewall solutions and service quality configuration. In voluntary cooperation models, service agreements cannot be made in the same way, since both parties have responsibilities to one another.

We do not consider the details of the negotiation process here, since it is potentially quite complicated; rather we shall assume, for this work, that a negotiation has taken place. The result of this process is a list of logical neighbours who are considered to be 'method peers', i.e., peers who will execute remotely requested methods.

Service Provision Phase

After intentions have been established, one needs a way of safely exchanging data between contracted peers in such a way that neither party can subsequently violate the voluntary trust model. To implement this, we demand that a client cannot directly solicit a response from a server. Rather, it must advertise to trusted parties that it has a request and wait to see if a server will accept it of its own free choice.

1. Host A: Advertises a service to be carried out, by placing it in a public place (e.g., virtual bulletin board) or by broadcasting it.
2. Host B: Scout A looks to see if host A has any jobs to do, and accepts job. Host B advertises the result when completed by putting the result in a public place.
3. Host A: looks to see if host B has replied.

Seen from the perspective of host A and B, this is done entirely using 'pull' methods, i.e., each host collects the data that is meant for it; no host can send information directly to another, thereby placing demands on the other's resources. The algorithm is a form of batch processing by posting to local 'bulletin boards' or 'blackboards'. There is still risk involved in the interaction, but each host has finer control over its own level of security. Several levels of access control are applied:

- First the remote client must have permission to signal the other of its presence
- Next the server must approves the request for service and connect to the client.
- The client must then agree to accept the connection from the server (assuming that it is not a gratuitous request) and check whether it has solicited the request.

The basic functionalities of this approach may be recognized as typical broker or marketplace approaches for agents in a multi agent system. This makes the discussion particularly interesting because it addresses a relatively known scenario. Before we go into into the administrative challenges of this approach, we present one concrete implementation as a protocol model simulation.

Protocol Specification in Maude

The programming language Maude [28] is specializes in the simulation of distributed systems. In contrast to typical programming approaches, this language is used to state the allowed transitions of a configurations state to another. This opens for searches in the execution domain of the entire system. With its roots in transitional logic it has been used to analyze protocol related to mobile systems and security. The Maude framework enables the writer to focus on the essence in the protocol without wasting to much code on unrelated features. The process is especially helpful if one wants to examine prototype implementations of a protocol, for testing. A secondary effect is that formal specifications of a system often reveal hidden assumptions in the design of the protocol, which lead to differences in the protocol implementation and bugs.

Programming in Maude consists mainly of two tasks: (1) defining the data model in terms of classes and types of messages between objects and (2) defining the allowed transitions from one program state to another. It is thus declarative. Thus, instead of programming a particular path of execution, one sets the scene for all possible steps that might occur without assuming what happened before. Execution of a Maude specification is like rewriting logic.

If one has a starting configuration and a number of independent rewriting rules, one could, in principle, use any of the possible rules. Maude creates a tree, where the initial configuration is the root node and all possible outcomes form the rest. For complex systems it is clear that the human mind can imagine only few of all the possible configurations. Maude possesses a search mechanism to actually traverse this tree to look for certain configurations.

Maude takes an object-oriented approach when designing distributed systems. One usually defines 'class objects' or data-types for the objects that are to communicate with each others, and a class for messages. Below is an example of the classes specified in this implementation: A client (the node requesting service), a server, and a blackboard.

```
class Client | pendingRPC : RPCList,
 knownServers : OidList,
 jobs : JobList,
 completedJobs : JobList,
 rpcEnabled : OidList,
 blackBoard : Oid .

class Server | rpcClients : OidList,
 blackBoard : Oid .

class BB | rpcRequests : RPCList,
 rpcReplies : RPCList .
```

To illustrate the specification of a transitional rule, consider the following:

```
crl [accept-service-request] :
 < S : Server | rpcClients : OL >
 (msg RPCrequestService(C,S) from C to S )
 =>
 < S : Server | rpcClients : OL :: C >
 (msg RPCrequestReply(C,S,true) from S to C )
 if ( not containsElement(OL, C) ) .
```

The "=>" marks the transition from the state before to the state after. The state before says only that

there is a Server object S and a message addressed to that object. The server object (enclosed in "< >") has a list of object pointers called rpcClients. The client is added to that list if it is going to be served by the server. Note that Maude assumes that there may be many other objects currently in the same state. The only thing that is specified is that, if at a given point a server S sees the message, it may react to it. Nothing is stated about time constraints or priorities of transitions.

The displayed rule has a conditional that states that if a server sees a message addressed for it and containing a RPCrequestService(C,S) payload from client C it will answer with a reply but only if the client is not served by the server already. In this particular case it will answer "yes" to the request. A similar rule exists where the server answers "no". Maude chooses one of these rules randomly in its simulation.

Execution of a Maude program means supplying an initial state (also called Configuration) and letting Maude choose a path at random, based on the all the possible transitions of the given state. There is also the possibility of issuing searches in the execution tree (all possible paths) of the initial state in order to seek out undesirable states of the protocol. This is beneficial for system designers who work with distributed systems too complex for unaided humans to think of, in every possible outcome. Other fields of computer science have used formal modelling of this kind for a long time. We perceive this tool to be of value in the field of theoretical system administration as well.

Every execution tree is based on an initial configuration. To begin with something basic, lets take the case of one of client and one server. We define simpleInit to be a function that returns the following configuration:

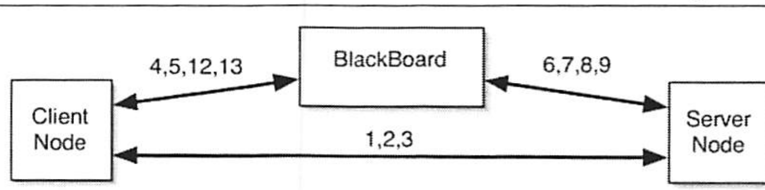
```
*** A simple initial state
eq simpleInit =
< "client" : Client | completedJobs : empty,
                 pendingRPC : nil,
 blackBoard : "BB",
 rpcEnabled : none,
         knownServers : "server",
         jobs : "job1" ^ "job2" >
< "server" : Server | blackBoard : "BB",
 rpcClients : none >
< "BB" : BB | rpcReplies : nil,
 rpcRequests : nil > .
```

All participants know about each other out-of-band, and so they start with variables pointing to each other. The client has two jobs it needs to be performed, "job1" and "job2". It has an empty list called completedJobs and the most basic search is to query whether this list can contain both jobs in the end. That would be a sign that the specification works as intended:

```
(search [1] simpleInit =>+
 < "client" : Client |
 completedJobs : "job1" ^ "job2",
 Client:AttributeSet >
 C':Configuration .
)
```

The search gives the desired result, which indicates that the protocol can indeed be used. But the main goal of this implementation was to review any hidden assumptions in the design. After implementing the protocol in Maude, several unaddressed assumptions became clear:

- A Job is an atomic object that is translated into a single RPC call and does not depend on other Jobs or on a sequential execution of other jobs.
- The service agreement request is a part of the actual RPC service and not a standalone service.
- A client node knows the address of blackboard and server node beforehand.
- A server node knows the address of the blackboard. (The blackboard could be a local process on one of the participants to maintain complete autonomy, as in the cfengine implementation.)
- Server and client node use the same blackboard in the simulation, but this sacrifices complete autonomy by introducing a third party. (This is fixed in the cfengine implementation.)
- No explicit mention is made about the duration of the service agreement. The server will agree to serve the client indefinitely.
- A client has no way of ending a relationship with a server.
- We assume a trusted transmission medium. No lost or doubled messages can affect the result.



1. [send-service-request]
2. [accept-service-request]
3. [recieve-service-agreement]
4. [send-RPC-request-to-bb]
5. [recieve-RPC-request-from-client]
6. [server-send-pullJob]
7. [bb-respond-to-pullJob]
8. [bb-drops-pullJob]
9. [server-does-job]
10. [blackboard-saves-reply]
12. [blackboard-returns-reply]
13. [blackboard-drops-reply-no-finished-job]

Figure 3: The different transitions in the Maude implementation of the protocol. The numbers represent the order of the transitions for a successful RPC call and reply.

Many of these assumptions are usually designed into RPC middleware, in order to provide transparency of these details.

So, after this analysis, suppose we go back to the vision of wireless mobile devices that engage with each other using a protocol like the one described. What are the ramifications to the network and service level?

Administrative Costs

A RPC call will in the Maude implementation of the protocol requires least 8 message components to be sent if there are no redundant pull messages from either side. The frequency of pull messages from both client and server is not specified in the protocol itself. A smaller interval may saturate the network, and use more battery capacity in a mobile device, but gives better service.

If we assume an equal transmission time t for a message between each participant and the pull-interval of i and a time S for a server to process the job and that all other processing of messages is close to instantaneous, we see that the smallest possible amount of time for a RPC call is: $t * 8 + S$.

Also the expected number of messages can be described as: $\frac{S}{i} + 8$.

The term *messages* is used explicitly, because it may not correspond to the a single packet per message. It may be so for the pull and service request messages, since they would be small, but for the job and the result, there are no clear limits on how small or big they may be. One should also point out, that the messages, which constitute the "overhead" compared to a direct service, are also the smallest ones. The job and the result would have to be transmitted regardless of the protocol and should not be a measure of its efficiency. What may be noted, is that they travel twice in this protocol. So for a shared medium network, they will occupy the medium twice as long.

Early in the text we pointed out the problem of current client-server technologies with regard to uncontrollable pressure on servers and the risk of bottlenecks and DoS-like saturations. Voluntary cooperation is proposed as a possible way of reducing this risk, although the negotiation phase of the protocol still contains direct traffic between the client and server. The less discussed participant in the plot, however, is the blackboard. It will receive more traffic and data then any of the two nodes that actually are using the service. If one looks at the administrative ramifications of this, one has in practice created a proxy which will get more network pressure than a typical client-server scenario would. Also, there is no voluntary cooperation in the blackboards part. Upon receiving a RPCjob it will have to store the RPCreply too at a later point.

A cfengine Implementation

The voluntary RPC has been implemented as a cfengine collaborative application. Remote services are referred to one another as "methods" or function calls. An automated negotiation mechanism has not been implemented as this remains an open question. We begin by assuming that a negotiation has taken place that determines which hosts will be clients and which will be servers and what the nature of the service is. This establishes each hosts "Method Peers," i.e., the hosts it is willing to provide services for. On the server host the update.conf rules must also contain a list of hosts to check for pending method requests.

```
MethodPeers = ( client1 client2 )
MethodPeers = ( GetPeers(*) )
```

The client *and* server hosts must then have a copy of the invitation:

```
methods:

  client_host||server_host::

  MethodExample("Service request",parameter)

  action=cf.methodtest
  server=server_host
  returnclasses=ready
  returnvars=retval

  ifelapsed=120
  expireafter=720

alerts:

  MethodExample_ready::

  "Service returned: $(MethodExample.retval)"
```

The predicate classes in the line ending in double colons tells the software that this method stanza will be read both by the client and the server host. On the client it is a request, and on the server it is the invitation. The action and server attributes determine where the methods are defined and who should execute them. Return values and classes are declared serving as access control lists for returned parameters. Since a remote method is not under out local control, we want to have reasonable checks about the information being returned. Methods could easily be asked to return certificates, for instance, to judge their authenticity. The ifelapsed and expireafter attributes apply additional scheduling policy constraints, as mentioned above.

On server host a copy of the and invitation is needed to counter-sign the agreement. The module itself must then be coded on the server. The invitation is based on the naming of the hosts; this has caused some practical problems due to the trust model. The use of IP addresses and ties to domain names has proven somewhat unreliable due to the variety of implementations of Domain Name Service resolution software. A name might be resolved into an IPv6 address on one end of a connection, but as an IPv4 address on the other side. This leads to mismatches

that are somewhat annoying in the current changeover period, but which are not a weakness of the method in principle. See Display 1.

A notable point regarding this implementation is that the protocol is incorporated into an existing service. This limits the range of possible applications, but reduces extra overhead since the RPC calls and results are bundled with the already scheduled communication.

Incorporation of Voluntary RPC in cfengine

The challenge of voluntary cooperation is to balance a regularity of service (constraints on time) with the need for security against misuse attacks. The reliability of services becomes a matter for a host's reputation, rather than a matter of contract. Hosts must learn each others' probable behaviour over time rather than demand a certain level of service. Clearly this is not acceptable in every situation, but it is a desirable way of opening relations with a host that one has never seen before.

Given that all service execution is based on voluntary cooperation, there are no guarantees about service levels. This is a feature of the protocol. However, this does not mean that arbitrary bounds on service levels are not implementable. Cooperative hosts can easily arrange for a rapid level of service by agreeing on a schedule for contacting their peers. This can be as often as each party feels is appropriate.

If we assume that all host agents run diagnostics and administrative services with the same regular period P, and that all servers are willing to process the request without delay, then we can say that the expected time to service $\langle T \rangle$ is:

$$\langle P \rangle \leq \langle T \rangle \leq 2\ \langle P \rangle .$$

This may be verified empirically. Limits can also be placed on the maximum number of times that a method can be evaluated in a certain interval of time.

Note that the scheduling period P can be arbitrarily small, but for administrative activities one is usually talking about minutes rather than milliseconds. The efficiency of the method for small P is not impressive, due to the overheads of processing the additional steps. Thus the voluntary RPC is not suggested as a replacement for conventional service provision.

In the cfengine software implementation, the locking parameters 'ifelapsed' and 'expireafter' determine additional controls that bind the evaluation of remote methods tightly to policy. The 'ifelapsed' time says that a method will be ignored until the specified time has elapsed since the last execution. The 'expireafter' option says that a still-executing method will not be allowed to continue for longer than the expiry time.

Discussion and Comments

The implementation in cfengine does not contain a negotiation phase as a part of the RPC protocol. As implemented, cfengine assumes that type of service and its parameters is established out-of-band. Also, there is no direct reference to any blackboard that caches the RPC interaction and thus there is complete autonomy.

```
control:

 MethodName = ( MethodExample )
 MethodParameters = ( value1 value2 )
 MethodAccess = ( patterns )

 # value1 is passed to this program, so lets
 # add to it and send the result back for fun

 var1 = ( "${value1}...and get it back" )

 actionsequence = ( editfiles )

###############################################

classes:

 moduleresult = ( any )
 ready = ( any )

###############################################

editfiles:

{ /tmp/file1

AutoCreate
AppendIfNoSuchLine "Important data...$(value2)"
}

###############################################

alerts:

 moduleresult::

 ReturnVariables("${var1}")
 ReturnClasses(ready)
```

Display 1: Extended invitation example.

A more finely grained response regulation and access control is implemented in cfengine as compared to the more translucent concept of a "job" in the Maude implementation. There is little leeway for the nodes to change any of these parameters itself in any of the implementations and may therefore seem as not very dynamic or adaptable. It is also left to the implementation to sort out special cases, such as if the server would not want to serve the client but the RPC call is on the blackboard. Cfengine simply ignores such cases and clears them up as garbage. What coordination and messages would be necessary in order to remedy this situation?

The implementation in cfengine raises an important question: if this were a truly pervasive environment and the nodes just got to know each other, how would they agree on the parameters and correct return type of the RPC? Why is this important? Because if the negotiation takes more computing power than the actual service in question then what is the gain for a mobile node to participate? Seen as a game, what is the cost-benefit relationship for the client and server? One has to think of the wider context. We must try to keep in mind future services that nodes may offer to each other, where such a negotiation is worth the delays.

Based on our observations, it is natural to think of a realistic future scenario as being partially mobile, perhaps with some fixed infrastructure. that may serve as a cache for blackboard messages. For example, in a virtual shopping mall, a blackboard service might be provided, taking its own share of the risk from customers. Nodes have greater freedom through this protocol: they may actually be away from the network while the contents are cached by the blackboard. A global ID, like the MIPv6 protocols home address, will make sure that the nodes can gather the data belonging to it from anywhere. It must be noted also, that the blackboard does not have to be a single hop away. It can be several, making this service more global and not just limited to a single network.

So, is there really any need for this type of voluntary cooperation?

One model of pervasive services is commercial and based on the Internet café. Here roaming tourists pay for a ticket that grants them access to an otherwise closed local service. In regular commerce, it is assumed that this monetary relationship is sufficient to guarantee a decent form of behaviour, in which parties behave predictably and responsibly. But is this necessarily true? There are several faults in this assumption for network services.

First of all, the commerce model assumes that virtual customers will behave much as they have done in human to human commerce. There is no compelling reason to suppose that this will be the case. A generation of users has now grown up taking lawless online technologies for granted. Technologies such as mobile phones, peer to peer sharing and Internet chat rooms are *changing* societal attitudes and the rules of interaction between young people [29].

Secondly, the model assumes that clients will exhibit predictable behaviour. Studies show that such a predictable relationship must be built up over time, and only applies if customers are regular patrons of a service provider [30]. Customer trust is based on the idea that customers and providers form relatively long lasting relationships and that one can therefore punish misdeeds by some kind of tit-for-tat interaction [8], because they will almost certainly meet again in the future. However, in a future where everything is on the move, distance has no meaning, and both children and adults are very much on-line, there is reason to suppose that clients will not 'hit and run' servers for prank or for malice. Hiding or changing identity on the Internet is trivial, so customers can visit providers repeatedly with a new identity and there avoid reprisals.

Finally, the use of of immediate payment as a barrier to deter frivolous usage assumes users will behave rationally in an adult fashion. This assumption is also flawed, since users might expect their uncooperative actions to have greater payoffs down the line. All of this points to the need for greater initial caution, and a reward scheme for cooperative behaviour.

A simple protocol for voluntary cooperation has shown that with a slightly higher load on a network one can reduce the risks of connecting with untrusted clients. If these types of services arrive, using independent blackboards (e.g., Grid Computing), the system administrators may see the dawn of a new type of servers which act only as caches and proxies for voluntary interaction. They will in turn be the ones that need protection from saturation and abuse. If fully autonomous behaviour is preserved, these problems can be avoided. We are currently working on a theory of fully autonomous interaction called promise theory [31].

Is voluntary cooperation a technical challenge or a human challenge then? The answer must be that it is both. A device that is misappropriated for malicious purposes by a human behaves simply as a hostile device to other computers. Computers perform a useful service to maintain the integrity of the networked community if they try to protect themselves, regardless of the source of the mischief.

Conclusions

This paper discusses the issue of voluntary cooperation in peer networks, and analyses a proposed trade-off which might help to lower the risks of client-server interactions in unpredictable environments.

The mechanism proposed is one of minimal acceptable trust. Through the mechanism, hosts in a collective environment can maintain maximal administrative control of their own resources, and hence reduce the risk of abuses and security breaches. Voluntary

cooperation (or pull-based service provision) is fully implementable and offers flexibility of asynchronous timing, as well as some additional security.

The motivation for this work was initially to support an administrative peer to peer algorithm that respects the autonomy of individual peers (inspired by model 6 of refs. [13, 3]), but the result is of more general interest. Its application domain is mainly tasks that are the traditional domain of the system administrator, but which have to operate in an uncertain and far from controlled environment.

A simulation in the analysis framework Maude, and an test implementation in the configuration tool cfengine [32, 33], show that the protocol functions properly, as advertised, and reveals the nature of the tradeoffs. The protocol overhead can be reduced, if voluntary cooperation is incorporated into an existing service, as in cfengine.

We have tested the voluntary cooperation approach on mundane tasks like the backup of remote machines, distributed monitoring (exchange of usage data, confirmation of uptime etc.), load balancing and other tasks. Although this mechanism currently has a limited number of desirable applications, this is probably due, in part, to the lack of truly mobile administrative approaches to hosts, in contemporary organizations. As pervasive services mature, one could expect that a voluntary cooperation mechanism would actually become invaluable for high risk transfers, e.g., in distributed software updating.

For demand services, the minimal trust model seems both cynical and inappropriate: perhaps even a hindrance to cooperative behaviour and commerce. At some point, the actors of the virtual community online must learn the human rules of trusting engagement, and new generations of children will have to be educated to accept that virtual communities are just as important as real ones. Human relations are based on rules of voluntary etiquette, but such social niceties have taken thousands of years to evolve.

We present this work in the framework of system administration because system administrators are the virtual custodians of the hosts and devices in the pervasive computing scenario. If administrators do not discuss the issues of scalable and management, and devise a workable scenario in the face of all its uncertainties, no one will.

Acknowledgement

We are remotely grateful to Siri Fagernes and John Sechrest for voluntary discussions on cooperative services.

Bibliography

[1] Sapuntzakis, C. and M. S. Lam, "Virtual appliances in the collective: A road to hassle-free computing," *Proceedings of the Ninth Workshop on Hot Topics in Operating Systems (HOTOS IX)*, 2003.

[2] Begnum, K., M. Burgess, and J. Sechrest, "Infrastructure in a virtual grid landscape from abstract roles," *Journal of Network and Systems Management*, (submitted).

[3] Burgess, M., and G. Canright, "Scaling behaviour of peer configuration in logically ad hoc networks," *IEEE eTransactions on Network and Service Management*, Vol. 1, Num. 1, 2004.

[4] Burgess, M., and S. Fagernes, "Pervasive computing management: A model of network policy with local autonomy," *IEEE eTransactions on Network and Service Management*, submitted.

[5] Burgess, M., and S. Fagernes, "Pervasive Computing Management II: Voluntary Cooperation," *IEEE eTransactions on Network and Service Management*, submitted.

[6] Burgess, M., and S. Fagernes, "Pervasive computing management III: Management Analysis," *IEEE eTransactions on Network and Service Management*, submitted.

[7] Undercoffer, J., F. Perich, A. Cedilnik, L. Kagal, and A. Joshi, "A secure infrastructure for service discovery and access in pervasive mputing," *Mobile Networks and Applications*, Vol. 8, Num. 2, pp. 113-125, 2003.

[8] Axelrod, R., *The Evolution of Co-operation*, Penguin Books, 1984.

[9] Martin-Flatin, J. P., "Push vs. pull in web-based network management," *Proceedings of the VI IFIP/IEEE IM conference on network management*, p. 3, 1999.

[10] Burgess, M., "Computer Immunology," *Proceedings of the Twelth Systems Administration Conference (LISA XII)*, p. 283, USENIX Association, Berkeley, CA, 1998.

[11] Paxson, V. and S. Floyd, "Wide area traffic: the failure of poisson modelling," *IEEE/ACM Transactions on networking*, Vol. 3, Num. 3, p. 226, 1995.

[12] Bellavista, P., A. Corradi, R.Montanari, and C. Stefanelli, "Policy-driven binding to information resources in mobility-enabled scenarios," *Mobile Data Management, Lecture Notes in Computer Science*, Num. 2574, pp. 212-229, 2003.

[13] Burgess, M., and G. Canright, "Scalability of peer configuration management in partially reliable networks," *Proceedings of the VIII IFIP/IEEE IM conference on network management*, p. 293, 2003.

[14] Anderson, E., M. Burgess, and A. Couch, *Selected Papers in Network and System Administration*, J. Wiley & Sons, Chichester, 2001.

[15] Fradet, P., V. Issarny, and S. Rouvrais, "Analyzing non-functional properties of mobile agents,"

Fundamental Approaches to Software Engineering, Lecture Notes on Computer Science, Num. 1783, pp. 319-333, 2000.

[16] Case, J., M. Fedor, M. Schoffstall, and J. Davin, "The simple network management protocol," *RFC1155*, STD 16, 1990.

[17] Zapf, M., K. Herrmann, K. Geihs, and J. Wolfang, "Decentralized snmp management with mobile agents," *Proceedings of the VI IFIP/IEEE IM conference on network management*, p. 623, 1999.

[18] Matsushita, M., "Telecommunication management network," *NTT Review*, Num. 3, pp. 117-122, 1991.

[19] Sloman, M. S. and J. Moffet, "Policy hierarchies for distributed systems management," *Journal of Network and System Management*, Vol. 11, Num. 9, p. 1404, 1993.

[20] Lignau, Anselm, Jürgen Berghoff, Oswald Drobnik, and Christian Mönch, "Agent-based configuration management of distributed applications," *Proceedings of the 3rd International Conference on Configurable Distributed Systems*, pages 52-59, 1996.

[21] Oram, Andy, editor, *Peer-to-peer: Harnessing the Power of Disruptive Technologies*, O'Reilly, Sebastopol, California, 2001.

[22] Dunbar, R., *Grooming, Gossip and the Evolution of Language*, Faber and Faber, London, 1996.

[23] Lee, M. K., and X. H. Jia, "A reliable asynchronous rpc architecture for wireless networks," *Computer Commmunications*, Vol. 25, Num. 17, pp. 1631-1639, 2002.

[24] Kottmann, D., R. Wittmann, and M. Posur, "Delegating remote operation execution in a mobile computing environment," *Mobile Networks and Applications*, Vol. 1, Num. 4, pp. 387-397, December, 1996.

[25] Bakre, Ajay V., and B. R. Badrinath, "Reworking the RPC paradigm for mobile clients," *Mobile Networks and Applications*, Vol. 4, pp. 371-385, 1997.

[26] Burgess, M., G. Canright, and K. Engø, "A graph theoretical model of computer security: from file access to social engineering," *International Journal of Information Security*, Vol. 3, pp. 70-85, 2004.

[27] Watts, D. J., *Small Worlds*, Princeton University Press, Princeton, 1999.

[28] Maude, *The maude homepage*, http://maude.cs.uiuc.edu .

[29] Rheingold, Howard, *Smart Mobs: The Next Social Revolution*, Perseus Books, 2002.

[30] Axelrod, R., *The Complexity of Cooperation: Agent-based Models of Competition and Collaboration*, Princeton Studies in Complexity, Princeton, 1997.

[31] Burgess, Mark, "An approach to understanding policy based on autonomy and voluntary cooperation," *IFIP/IEEE 16th international workshop on distributed systems operations and management (DSOM), in LNCS 3775*, pages 97-108.

[32] Burgess, M., "A site configuration engine," *Computing systems*, Vol. 8, p. 309, MIT Press, Cambridge, MA, 1995.

[33] Burgess, M., *Cfengine www site*, http://www.iu.hio.no/cfengine, 1993.

Network Configuration Management via Model Finding

Sanjai Narain – Telcordia Technologies, Inc.

ABSTRACT

Complex, end-to-end network services are set up via the configuration method: each component has a finite number of configuration parameters each of which is set to a definite value. End-to-end network service requirements can be on connectivity, security, performance and fault-tolerance. However, there is a large conceptual gap between end-to-end requirements and detailed component configurations. To bridge this gap, a number of subsidiary requirements are created that constrain, for example, the protocols to be used, and the logical structures and associated policies to be set up at different protocol layers.

By performing different types of reasoning with these requirements, different configuration tasks are accomplished. These include configuration synthesis, configuration error diagnosis, configuration error fixing, reconfiguration as requirements or components are added and deleted, and requirement verification. However, such reasoning is currently ad hoc. Network requirements are not even precisely specified hence automation of reasoning is impossible. This is a major reason for the high cost of network management and total cost of ownership. This paper shows how to formalize and automate such reasoning using a new logical system called Alloy.

Alloy is based on the concept of model finding. Given a first-order logic formula and a domain of interpretation, Alloy tries to find whether the formula is satisfiable in that domain, i.e., whether it has a model. Alloy is used to build a Requirement Solver that takes as input a set of network components and requirements upon their configurations and determines component configurations satisfying those requirements.

This Solver is used in different ways to accomplish the above reasoning tasks. The Solver is illustrated in depth by carrying out a variety of these tasks in the context of a realistic fault-tolerant virtual private network with remote access. Alloy uses modern satisfiability solvers that solve millions of constraints in millions of variables in seconds. However, poor requirements can easily nullify such speeds. The paper outlines approaches for writing *efficient* requirements. Finally, it outlines directions for future research.

Introduction

Complex, end-to-end network services are set up via the configuration method: each component has a finite number of configuration parameters each of which is set to a definite value. End-to-end network service requirements can be on connectivity, security, performance and fault-tolerance. However, there is a large conceptual gap between end-to-end requirements and detailed component configurations. To bridge this gap, a number of subsidiary requirements are created that constrain, for example, the protocols to be used, and the logical structures and associated policies to be set up at different protocol layers.

By performing different types of reasoning with these requirements, different configuration tasks are accomplished. These include configuration synthesis, configuration error diagnosis, configuration error fixing, reconfiguration as requirements or components are added and deleted, and requirement verification. However, such reasoning is currently ad hoc. Network requirements are not even precisely specified hence automation of reasoning is impossible. This is a major reason for the high cost of network management and total cost of ownership.[1] This paper shows how to formalize and automate such reasoning using the new concept of a Requirement Solver, as shown in Figure 1.

This Solver takes as input a set of network components and requirements upon their configurations and

[1]“. . . operator error is the largest cause of failures . . . and largest contributor to time to repair . . in two of the three (surveyed) ISPs . . configuration errors are the largest category of operator errors.” [1]

“Although setup (of the trusted computing base) is much simpler than code, it is still complicated, it is usually done by less skilled people, and while code is written once, setup is different for every installation. So we should expect that it's usually wrong, and many studies confirm this expectation.” [2]

“Consider this: ..the complexity (of computer systems) is growing beyond human ability to manage it..the overlapping connections, dependencies, and interacting applications call for administrative decision-making and responses faster than any human can deliver. Pinpointing root causes of failures becomes more difficult.” [3]

computes as output, component configurations satisfying those requirements. The requirements can be in first-order logic (Boolean logic with quantifiers on individual variables). This logic is highly expressive and captures a very large class of practical network requirements.

The Requirement Solver is used in different ways to accomplish the above reasoning tasks. The Solver is illustrated in depth by carrying out a variety of these tasks in the context of a realistic fault-tolerant virtual private network with remote access. The reasoning tasks are accomplished as follows:

1. **Configuration Synthesis.** To determine how to configure a set of components so they satisfy a system requirement R, submit the set of components and R to the solver and take the output.
2. **Requirement Strengthening.** If a set of components satisfies a system requirement R but must now satisfy another requirement S, then to reconfigure components, submit the set of components and R∧S to the solver and take the output.
3. **Component Addition.** If a new component is to be added to a set of components already satisfying requirement R, then to configure the new component and possibly reconfigure existing components, submit the new set of components and R to the Solver and take the output.
4. **Requirement Verification.** To prove that it is impossible for an undesirable requirement U to be true when a set of components satisfies requirement R, submit the set of components and R∧U to the Solver. If the Solver cannot find a solution, the assertion is proved. Otherwise, the Solver produces a counterexample.
5. **Configuration Error Detection.** To check whether configuration of a given set of components is consistent with a requirement R, represent the configuration as a set C of constraints each of the form P=V where P is a configuration parameter and V its value. Then, submit the set of components and R∧C to the solver. If the solver cannot produce a solution, a configuration error is detected.
6. **Configuration Error Fixing.** If the configuration of a given set of components is inconsistent with a requirement R, then submit the set of components and R to the Solver and find a new solution that is as "close" as possible to the current configuration.

The Requirement Solver has been inspired by the new logical system called Alloy [4]. While Alloy is based in set theory, a subset of it also has an intuitive object-oriented interpretation: it lets one specify object types, their attributes and type of attribute values. It also lets one specify first-order logic constraints on these. Finally, it lets one specify a "scope" that defines a *finite* number of object instances of each type in a given system.

Given a specification and a scope, Alloy attempts to find values of attributes of object instances in the scope that satisfy the specification. These values together constitute a "model" of the specification in the system in the logical sense of the word "model". Alloy first compiles a specification into a propositional formula in conjunctive normal form, then uses a satisfiability solver such as Berkmin [5] or Zchaff [6] to check whether the formula is satisfiable. If so, it converts satisfying values of propositional variables back into values of attributes and displays these.

Often, more than one solution is found. Even though satisfiability is NP-complete in the worst case, in the *average* case, solutions are efficiently found. Modern satisfiability solvers can solve millions of constraints in millions of variables in seconds. However, poor requirements can easily nullify such speeds. The paper outlines approaches for writing *efficient* requirements. Finally, it outlines directions for future research.

The Solver has a direct implementation in Alloy. Network component types, attributes and values are directly modeled in Alloy. A set of network components of different types is modeled as a scope. Network system configuration is modeled as values of all component attributes in a given scope. Network requirements are modeled using first-order logic. Solutions are found by the Alloy model finder.

Subsequent sections illustrate Alloy's capabilities, describe the design of a fault-tolerant VPN with remote access and the challenges of setting it up, outline a

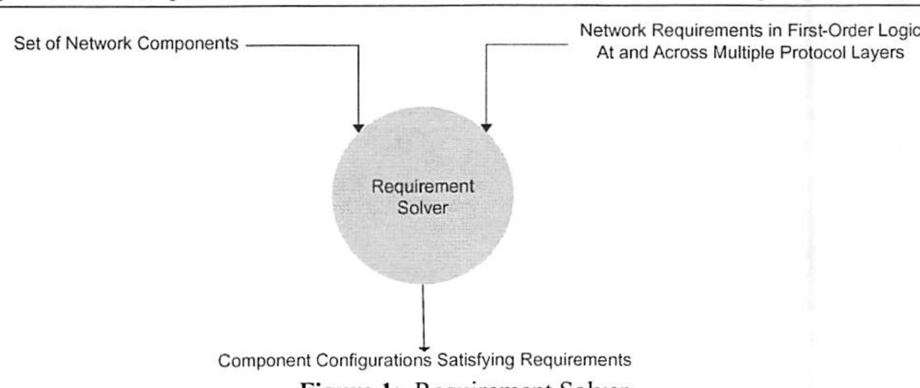


Figure 1: Requirement Solver.

formalization of the design in Alloy, describe how to accomplish, respectively, tasks 1-4 above, outline approaches for writing efficient requirements, outline relationship with previous work, summarize this work and the conclusions, and present directions for future research.

Alloy By Example

The three lines in Display 1 declare three object types: router, subnet and interface. The last type has two attributes, the chassis (of type router) to which it belongs, and network (of type subnet) on which it is placed. These attributes model configuration parameters of an interface.

The predicate spec in Display 2 defines a specification whose model we will try to find. It is a conjunction of three constraints. The first states that for every router x there is an interface y whose chassis is x, i.e., every router has at least one interface. The second states that no two non-equal interfaces on the same router are placed on the same subnet. The third states that our network contains one router, two subnets and two interfaces. These are the components we want to configure.

The Alloy command shown in Display 3 produces the model (values of configuration parameters) shown in Display 4. The first line states that the value of chassis for interface_0 is router_0 and the value of chassis for interface_1 is router_0. Similarly, for the second line. Alloy automatically creates instances of objects such as router_0, subnet_0, and subnet_1. Note that Alloy did not place interface_0 and interface_1 on the same subnet due to the second constraint. On the other hand, if we remove that constraint, Alloy produces the additional solution show in Display 5 in which both interface_0 and interface_1 are placed on the same subnet.

Fault-Tolerant Virtual Private Network With Remote Access

Our top-level goal is to synthesize a fault-tolerant network that enables hosts, including mobile hosts, at geographically distributed sites to securely collaborate. A network design for achieving this goal is now outlined. When this is implemented via configuration, one obtains a network of the type shown in Figure 2. The existence of a wide-area network, represented by the WAN router in the figure, is assumed.

Each site has a gateway router called a spoke router whose external (or public) interface is connected to the WAN and whose internal (or private) interface is connected to hosts and servers in the site. A routing protocol is run on the external interfaces of spoke and WAN routers to automatically compute routes between these interfaces. As traffic between hosts and servers on different sites is intended to be private, it cannot be allowed to flow directly over the wide area network. In order to secure it, one possibility is to set up a full-mesh of IPSec tunnels between gateway routers. However, full-mesh is not scalable since the number of tunnels increases as the square of the number of sites: for the 200 sites expected in our domain, the number of tunnels would be nearly 20,000.

A scalable alternative is a hub-and-spoke architecture as shown. A certain number of hub routers is set up. Each spoke router sets up an IPSec tunnel to each hub router. Traffic from one site to another goes via two tunnel hops, one from its spoke router to a hub router and another from the hub router to the destination site's spoke router. The number of tunnels now only increases linearly with the number of sites.

The problem, however, is that if a hub router fails, connectivity between sites is lost. This is because the

```
sig router {}
sig subnet{}
sig interface {chassis: router, network: subnet}
```

Display 1: Declaring three object types.

```
pred spec ()
      {all x:router | some y:interface | y.chassis = x}
      {no disj x1,x2:interface |
            x1.chassis=x2.chassis && x1.network = x2.network}
      {#router=1 && #subnet=2 && #interface=2}
```

Display 2: A specification with three constraints.

```
run spec for 1 router, 2 subnet, 2 interface
```

Display 3: Alloy command to create a model.

```
chassis : = {interface_0 -> router_0, interface_1 -> router_0}
network : = {interface_0 -> subnet_1, interface_1 -> subnet_0}
```

Display 4: Results of command in Display 3.

```
chassis : = {interface_0 -> router_0, interface_1 -> router_0}
network : = {interface_0 -> subnet_0, interface_1 -> subnet_0}
```

Display 5: Less constrained solution.

source spoke router will continue to send traffic through the IPSec tunnel to the failed hub router. IPSec has no notion of fault-tolerance that will enable the source spoke router to redirect traffic via another hub router.

Routing protocols such as RIP or OSPF accomplish precisely this kind of fault-tolerance, however they are incompatible with IPSec. They do not recognize an IPSec tunnel as directly connecting two routers since there can be multiple physical hops in between. The solution is to set up a new type of tunnel, called GRE, between each hub and spoke router. The purpose of GRE is to create the illusion to a routing protocol that two routers are directly connected, even when there are multiple physical hops in between.

This is done by creating new GRE interfaces at each tunnel end point and making these belong to the same point-to-point subnet. Now, if a hub router fails, a routing protocol will automatically direct traffic through another GRE tunnel to another hub router, and then to the destination. Each GRE tunnel is then secured via an IPSec tunnel. Thus, the required fault-tolerant virtual private network is set up. If two hub router failures are to be tolerated, then three hub routers are required. Then, the number of tunnels to be set up is just 600 (number of hub routers × number of spoke routers) or 3% of nearly 20,000 in the full mesh case.

This solution has a useful defense-in-depth feature: there are two separate routing domains, the external one and the overlay one. No routes are exchanged between these. Thus, even if an adversary compromises the WAN router, he cannot send a packet to a host. The WAN router does not even have a route to the host.

In order to enable remote users to securely collaborate, a remote access server is set up in "parallel" with a spoke router. A remote user connected to the WAN sets up an L2TP tunnel between his host and the server. This tunnel gives the illusion to the host of being directly connected to the internal interface of the sites' spoke router. Consequently, all traffic between the host and any other host or server on the VPN is also secured. Again, one has to ensure that two separate routing protocols run on the access server, one for the private side and one for the public. In order to realize the above design, the following types of configuration parameters need to be set:

1. Addressing: Router interface address, type and subnet mask.
2. IPSec: Tunnel end points, hash and encryption algorithms, tunnel modes, preshared keys and traffic filters.
3. OSPF: Whether it is enabled at an interface, and OSPF area and type identifiers.
4. GRE: Tunnel end points and physical end points supporting GRE tunnels.
5. Firewall: Policies at each site.
6. Remote access: Subnets to which remote access server interfaces belong and routing protocols enabled on these.

It is very hard to compute values of the above configuration parameters. The types of configuration errors that can arise are:

1. Duplicate IP addresses may be set up, or all interfaces on a subnet may not have the same type.
2. IPSec tunnels may be set up incorrectly. For example, the preshared key, hash algorithm, encryption algorithm, or authentication mode may be unequal at the two tunnel end points. Peer values may not be mirror images of each other. These errors can lead to loss of connectivity. If the wrong traffic filter is used, then sensitive data can be transmitted without being encrypted.
3. OSPF routing domain may be set up incorrectly, for example, it may not be enabled at a required interface or the area and type identifiers may be

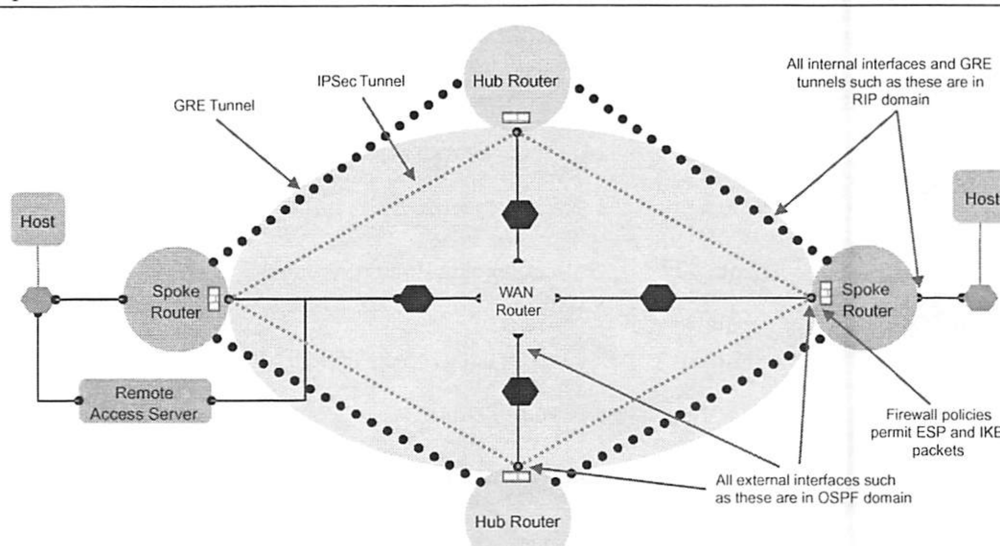


Figure 2: Fault-tolerant virtual private network with remote access.

incorrect. This can lead to incorrect routing tables and to outright isolation of subnets.
4. Routing loops may arise. If the same OSPF process is also used for routing between the gateway and WAN routers, then if it does not find a path through the physical network it will attempt to find a path through the overlay network. Since the overlay network is supported by the physical network, a routing loop will arise. This problem can be mitigated by using two distinct routing protocols, one for the overlay and another for the WAN.
5. GRE tunnels may be set up incorrectly. For example, the peer values may not be mirror images of each other, or the mapping between GRE ports and physical ports may be incorrect.
6. Firewall policies may block IPSec traffic, hence no traffic will pass through the tunnels.
7. Remote access interfaces may not belong to the correct subnets and incorrect routing protocols may be configured on these.

Before we show how to formalize the above design in Alloy, we capture its main intuitions in the following requirements:

- **RouterInterfaceRequirements**
  1. Each spoke router has internal and external interfaces
  2. Each access server has internal and external interfaces
  3. Each hub router has only external interfaces
  4. Each WAN router has only external interfaces
- **SubnettingRequirements**
  5. A router does not have more than one interface on a subnet
  6. All internal interfaces are on internal subnets
  7. All external interfaces are on external subnets
  8. Every hub and spoke router is connected to a WAN router
  9. No two non-WAN routers share a subnet
- **RoutingRequirements**
  10. RIP is enabled on all internal interfaces
  11. OSPF is enabled on all external interfaces
- **GRERequirements**
  12. There is a GRE tunnel between each hub and spoke router
  13. RIP is enabled on all GRE interfaces
- **SecureGRERequirements**
  14. For every GRE tunnel there is an IPSec tunnel between associated physical interfaces that secures all GRE traffic
- **AccessServerRequirements**
  15. There exists an access server and spoke router such that the server is attached in "parallel" to the router
- **AccessControlPolicyRequirements**
  16. Each hub and spoke external interface permits esp and ike packets

The interesting fact is these requirements do not specify the number of sites in the VPN. Rather, they apply to *all* sites. As new sites are added, these requirements are instantiated for the extended network to determine how to configure new components and reconfigure existing ones. This is a hard problem, in general, but that we show how to automatically solve with our approach.

Requirement Formalization In Alloy

This section presents an Alloy formalization of network component types, subtypes and their attributes. It also presents a formalization of Requirements 12 and 14. The complete formalization is available in the longer report at http://alloy.mit.edu/papers/NetConfigAlloy.pdf. Various types of routers are modeled using the following Alloy type declarations (signatures):

```
sig router {}
sig wanRouter extends router {}
sig hubRouter extends router {}
sig spokeRouter extends router {}
sig accessServer extends router {}
sig legacyRouter extends router {}
```

A generic interface just has a single attribute, the routing protocol enabled at it.

```
sig interface {routing:routingDomain}
```

A physical interface has two attributes, the router chassis on which it is mounted, and the network on which it is placed:

```
sig physicalInterface extends interface {
     chassis: router,
     network: subnet}
```

There are internal and external interfaces. External interfaces are of two types, one on hubs and one on spokes; see Display 6. There are two types of routing domains, RIP and OSPF:

```
sig routingDomain {}
sig ripDomain extends routingDomain {}
sig ospfDomain extends routingDomain {}
```

There are two types of subnets, internal and external:

```
sig subnet{}
sig internalSubnet extends subnet{}
sig externalSubnet extends subnet{}
```

There are three types of protocols, IKE, ESP and GRE:

```
sig internalInterface extends physicalInterface {}
sig externalInterface extends physicalInterface {}
sig hubExternalInterface extends externalInterface {}
sig spokeExternalInterface extends externalInterface {}
```

Display 6: Interfaces on hubs and spokes.

```
sig protocol {}
sig ike extends protocol {}
sig esp extends protocol {}
sig gre extends protocol {}
```

A firewall policy contains one of two possible permissions, permit and deny, that respectively, mean whether the firewall should allow a packet to go through or be dropped.

```
sig permission {}
sig permit extends permission {}
sig deny extends permission {}
```

A firewall policy, shown in Display 7, defines whether a packet associated with a protocol is allowed to go through or dropped, as it leaves an interface. An IPSec tunnel encrypts all packets associated with a protocol entering at either its local or its remote endpoint.

```
sig ipsecTunnel {
      local: externalInterface,
      remote: externalInterface,
      protocolToSecure: protocol}
```

A GRE tunnel encapsulates a packet into a new packet with source address that of its local endpoint and destination address that of its remote endpoint. Also, the tunnel is considered a proper link in a routing domain.

```
sig greTunnel {
      localPhysical: externalInterface,
      routing: routingDomain,
      remotePhysical: externalInterface}
```

An IP packet's attributes are its source and destination interfaces and the protocol it embodies. The precise data it carries is not modeled, since it is not relevant for our design purposes.

```
sig ipPacket {
      source:interface,
      destination:interface,
      prot:protocol}
```

Display 8 shows the Alloy version of Requirement 12. This states that between every hubExternalInterface x and spokeExternalInterface y there is a greTunnel whose local physical is x and remotePhysical is y, or vice versa.

Display 9 shows the Alloy version of Requirement 14. This states that for every greTunnel g there is an ipsecTunnel p that secures the gre protocol and whose endpoints are the same as the physical endpoints of g.

Configuration Synthesis

This section shows how to synthesize the initial network with connectivity and routing. Define:

```
PhysicalSpec =
      RouterInterfaceRequirements ∧
      SubnettingRequirements ∧
      RoutingRequirements
```

In Alloy, this would be expressed as:

```
Pred PhysicalSpec () {
      RouterInterfaceRequirements ()
      SubnettingRequirements ()
      RoutingRequirements ()}
```

Define a scope consisting of 1 hubRouter, 1 spokeRouter, 1 wanRouter, 1 internalInterface, 4 externalInterface, 1 hubExternalInterface, 1 spokeExternalInterface, 1 ripDomain, 1 ospfDomain, 3 subnet, 0 legacyRouter. These are the objects we want to configure. Now request Alloy to find a model for PhysicalSpec in the above scope. It synthesizes the network shown in Figure 3. It does so by producing the values of configuration parameters shown in Display 10. These are just the textual version of the network in Figure 3. Also note that spoke and hub routers are not directly connected, in accordance with Requirement 9.

Requirement Strengthening

In order to add an overlay network to the previous one, extend the previous scope with a GRE tunnel then request Alloy to satisfy (PhysicalSpec ∧ GRERequirements). Alloy synthesizes the network shown in Figure 4a. Alloy automatically sets up the GRE tunnel between the spoke and hub router and enables RIP routing on the GRE tunnel.

To make GRE tunnels secure, extend the previous scope with an IPSec tunnel and request Alloy to satisfy

```
sig FirewallPolicy {
       prot: protocol,
       action: permission,
       protectedInterface: physicalInterface}
```

Display 7: Firewall policy.

```
{all x:hubExternalInterface, y:spokeExternalInterface | some
g:greTunnel |
      (g.localPhysical=x && g.remotePhysical=y) or
      (g.localPhysical=y && g.remotePhysical=x)}
```

Display 8: Requirement 12 in Alloy.

```
{all g:greTunnel |
      some p:ipsecTunnel | p.protocolToSecure=gre &&
      ((p.local = g.localPhysical && p.remote = g.remotePhysical) or
       (p.local = g.localPhysical && p.remote = g.remotePhysical))}
```

Display 9: Requirement 14 in Alloy.

(PhysicalSpec ∧ GRERequirements ∧ SecureGRERequirements). Alloy synthesizes the network in Figure 4b. Alloy automatically places the IPSec tunnel between the correct physical interfaces to protect the GRE tunnel.

In order to add an access server to this network extend the previous scope with an access server, one internal interface, and one external interface and request Alloy to satisfy (PhysicalSpec ∧ GRERequirements ∧ SecureGRERequirements ∧ AccessServerRequirements). Alloy synthesizes the network in Figure 4c. Note that the access server is placed in parallel with only the spoke router, not with any other router, and has the correct routing protocols enabled on its interfaces.

Component Addition

When new components are added to an infrastructure, the logic that governs infrastructure has to be instantiated to the extended set of components. This is a nontrivial problem for humans to cope with. With Alloy, this instantiation is accomplished simply by finding a model of requirements for the existing scope extended by new components. (In order to avoid reconfiguring existing components, one can strengthen the requirement with existing configurations, each modeled as an equality P=V, P a configuration parameter and V its value). For example, in order to add a new spoke site to the previous network, extend its scope with a spoke

```
routing : =
    {externalInterface_0 -> ospfDomain_0,
     externalInterface_1 -> ospfDomain_0,
     hubExternalInterface_0 -> ospfDomain_0,
     internalInterface_0 -> ripDomain_0,
     spokeExternalInterface_0 -> ospfDomain_0}
chassis : =
    {externalInterface_0 -> wanRouter_0,
     externalInterface_1 -> wanRouter_0,
     hubExternalInterface_0 -> hubRouter_0,
     internalInterface_0 -> spokeRouter_0,
     spokeExternalInterface_0 -> spokeRouter_0}
network : =
    {externalInterface_0 -> externalSubnet_1,
     externalInterface_1 -> externalSubnet_0,
     hubExternalInterface_0 -> externalSubnet_0,
     internalInterface_0 -> internalSubnet_0,
     spokeExternalInterface_0 -> externalSubnet_1}
```

Display 10: Configuration parameters for Figure 3.

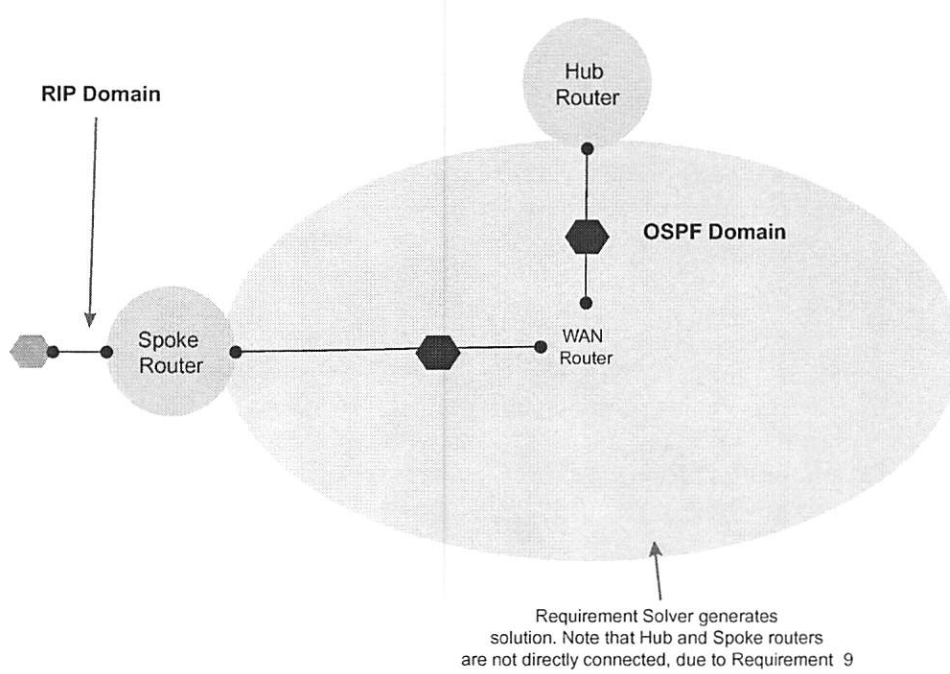


Figure 3: Configuration synthesis: Physical network.

```
(PhysicalSpec = RouterInterfaceRequirements ∧
                    SubnettingRequirements ∧ RoutingRequirements)
```

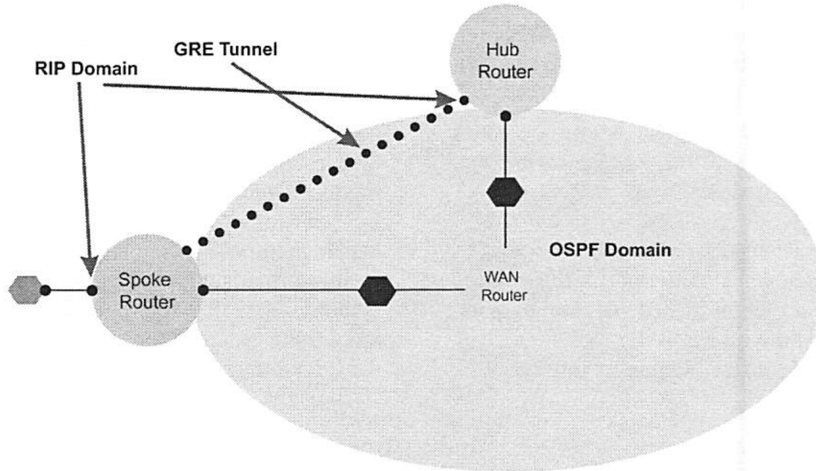


Figure 4a: Requirement strengthening: Adding overlay.
`(PhysicalSpec ∧ GRERequirements).`

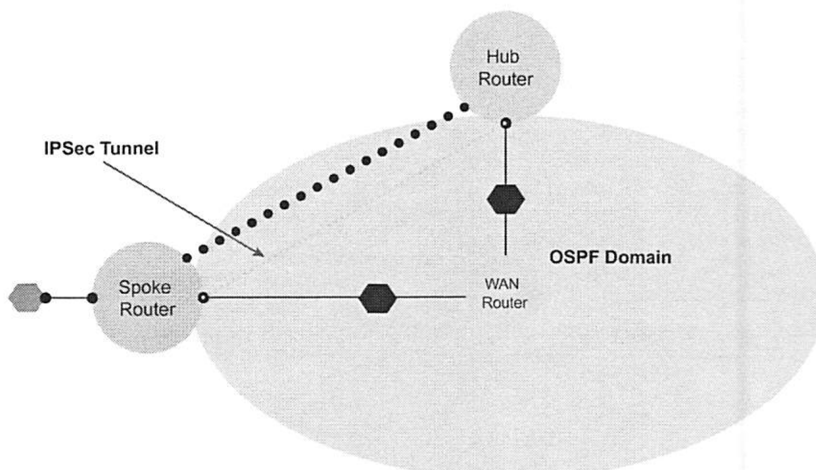


Figure 4b: Requirement strengthening: Securing overlay.
`(PhysicalSpec ∧ GRERequirements ∧ SecureGRERequirements)`

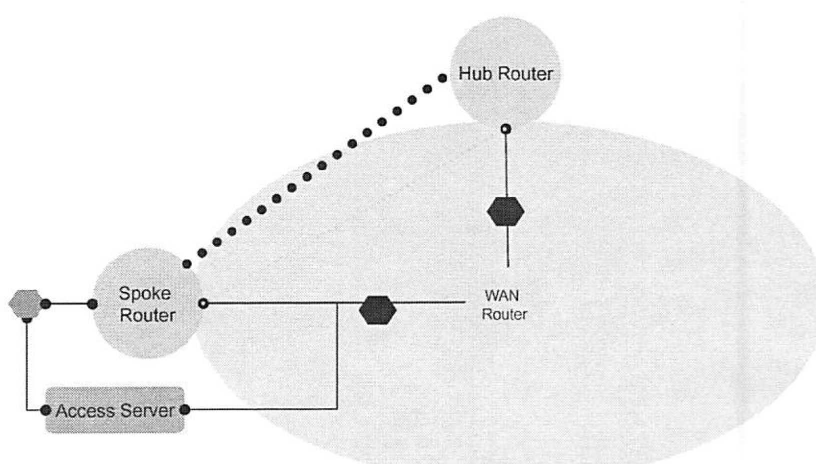


Figure 4c: Requirement strengthening: Adding remote access server.
`(PhysicalSpec ∧ GRERequirements ∧ SecureGRERequirements ∧ AccessServerRequirements)`

router, one internal subnet, one external subnet, one GRE tunnel and one IPSec tunnel. Requesting Alloy to synthesize a network satisfying (PhysicalSpec ∧ GRERequirements ∧ SecureGRERequirements ∧ AccessServer Requirements) in the new scope yields the network in Figure 5a.

Note that the new spoke router is physically connected just to the WAN router as required by Requirement 8. Moreover, GRE and IPSec tunnels are automatically set up between the new spoke router and hub router and physical interfaces and GRE tunnels are placed in the correct routing domains.

In order to add a new hub site to this network, extend its scope with a hub router, one external interface, one external subnet, two GRE tunnels and two IPSec tunnels. Requesting Alloy to synthesize a network satisfying (PhysicalSpec ∧ GRERequirements ∧ SecureGRE Requirements ∧ AccessServerRequirements) in the new scope yields the network in Figure 5b.

Finally, in order to permit IKE and ESP (protocols of IPSec) packets through the physical interfaces of hub and spoke routers, one can extend the above scope with eight firewall policies, then request Alloy to satisfy FullVPNSpec = (PhysicalSpec ∧ GRERequirements ∧ SecureGRERequirements ∧ AccessServerRequirements ∧ FirewallPolicyRequirements). Alloy then synthesizes the network of Figure 2 without the hosts. The reason for 8 firewall policies is that one policy is required for each IPSec tunnel endpoint.

Requirement Verification

Identifying Incorrect Firewall Policies

When we deployed the above network, we were careful to allow IKE and ESP packets to be permitted by access control lists at physical interfaces of hub and spoke routers. This was the reason for FirewallPolicyRequirements. However, we discovered that end-to-end connectivity was still not established. After considerable testing and analysis we realized that the WAN router itself was blocking IKE and ESP packets. We had not anticipated this cause. We now show how to formalize identification of this cause.

Alloy was not used for such identification, but this example illustrates how it could have been. Define a

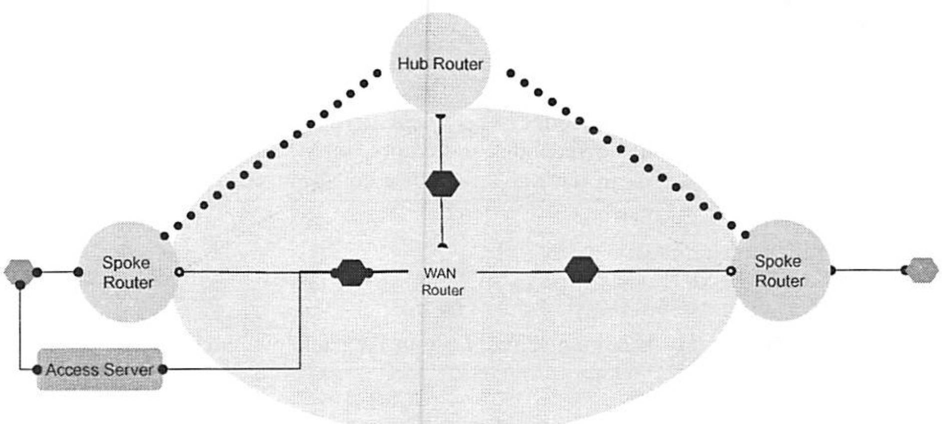


Figure 5a: Component addition: Adding new spoke router.
(PhysicalSpec ∧ GRERequirements ∧ SecureGRERequirements ∧ AccessServerRequirements)

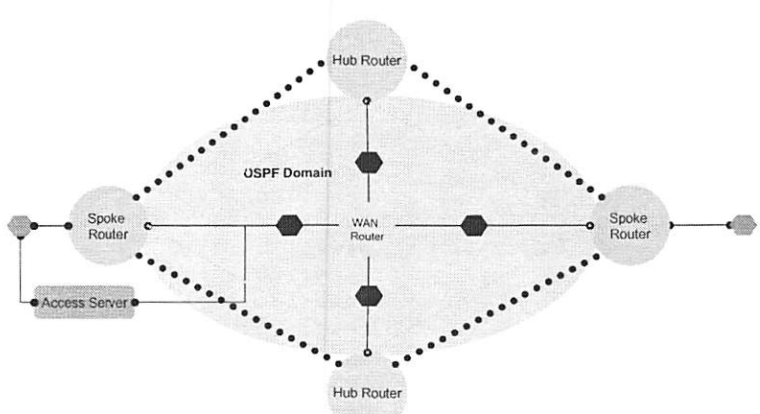


Figure 5b: Component addition: Adding new hub router.
(PhysicalSpec ∧ GRERequirements ∧ SecureGRERequirements ∧ AccessServerRequirements)

condition called BlockedIPSec capturing conditions under which an IPSec packet can be blocked, and find out how it is *possible* that (FullVPNSpec ∧ BlockedIPSec) be true. In other words, is it possible that the network be configured in a manner consistent with FullVPNSpec yet block IPSec packets? If so, we would have to modify requirements to preclude this possibility. The predicate in Display 11 states that IPSec is blocked if there is some esp or ike packet which is blocked. The predicate in Display 12 states that a packet is blocked if there is some firewall policy protecting an external interface that denies the protocol for that packet.

If we increase the scope of the last network to include 9 (more than 8) firewall policies, and request Alloy to find a model for (FullVPNSpec ∧ BlockedIPSec), Alloy produces the values in Display 13 for prot, permission and protectedInterface attributes of firewall policies. In other words, firewallPolicy_8, applied on externalInterface_0 on the WAN router, blocks ike_0.

Identifying Private Subnet Advertisement Of Private Subnets Into WAN

This section illustrates another problem that arose during deployment of our VPN solution into an existing network. Existing networks contain "legacy" routers that have no concept of internal or external subnets as do spoke routers. Thus, if our VPN is grafted into an existing network as shown in the figure above, the defense-in-depth feature mentioned in the second section is compromised. The legacy router can run the same routing protocol on both its internal and external interfaces and thereby export the internal subnet Y to the WAN. Now, if the WAN router is compromised, an adversary can send packets to the host at Y. We now show how to formalize identification of this possibility.

We define the code shown in Display 14. This predicate states that an internal subnet is advertised to the WAN if there is a legacy router with two interfaces, one attached to an internal subnet and another to an external subnet, and both have the same routing protocol enabled on them. Now, if we increase the scope of the network of Figure 5b to include one additional (legacy) router and two additional physical interfaces, and request Alloy to find a model for (FullVPNSpec ∧ internalSubnetAdvertisedToWan), Alloy produces the the code in Display 15. In other words, physicalInterface_0 and physicalInterface_1 can be placed on legacyRouter_0, one can be connected to an internal subnet, the other to an external subnet, and yet both can belong to ospfDomain_0.

Writing Efficient Requirements

Scope Splitting

One critical parameter to control in Alloy is the size of the scope. If it gets too large it should be split up and the specification changed, if necessary. Consider the following specification declaring router and interface types, and a relation chassis mapping an interface to its router. Also define EmptyCond to be an empty set of constraints to satisfy (Alloy requires *some* constraint before it can be run):

```
pred BlockedIPSec () {
some p:ipPacket, s,t:externalInterface |
        p.source = s && p.destination = t && (p.prot =
        ike or p.prot=esp) && Blocked(p)}
```

Display 11: Blocking IPSec if esp or ike packet is blocked.

```
pred Blocked(pack:ipPacket) {
        some p:firewallPolicy, x:externalInterface |
        p.protectedInterface = x &&
        p.prot=pack.prot &&
        p.action = deny
        }
```

Display 12: Block a packet if firewall policy denies its protocol.

```
prot : =
   {firewallPolicy_0 -> ike_0,
    firewallPolicy_1 -> ike_0,
    firewallPolicy_2 -> ike_0,
    firewallPolicy_3 -> ike_0,
    firewallPolicy_4 -> esp_0,
    firewallPolicy_5 -> esp_0,
    firewallPolicy_6 -> esp_0,
    firewallPolicy_7 -> esp_0,
    firewallPolicy_8 -> ike_0}
permission: =
   {firewallPolicy_0 -> permit_0,
    firewallPolicy_1 -> permit_0,
    firewallPolicy_2 -> permit_0,
    firewallPolicy_3 -> permit_0,
    firewallPolicy_4 -> permit_0,
    firewallPolicy_5 -> permit_0,
    firewallPolicy_6 -> permit_0,
    firewallPolicy_7 -> permit_0,
    firewallPolicy_8 -> deny_0}
protectedInterface : =
   {firewallPolicy_0 -> spokeExternalInterface_1,
    firewallPolicy_1 -> spokeExternalInterface_0,
    firewallPolicy_2 -> hubExternalInterface_1,
    firewallPolicy_3 -> hubExternalInterface_0,
    firewallPolicy_4 -> spokeExternalInterface_1,
    firewallPolicy_5 -> spokeExternalInterface_0,
    firewallPolicy_6 -> hubExternalInterface_1,
    firewallPolicy_7 -> hubExternalInterface_0,
    firewallPolicy_8 -> externalInterface_0}
```

Display 13: Model for FullVPNSpec ∧ BlockedIPSec.

```
sig router {}
sig interface {chassis: router}
pred EmptyCond () {}
```

When Alloy tries to find a model for EmptyCond in a scope consisting of 50 routers and 50 interfaces it crashes! This is because the cross product of the set of all routers and chassis' has 50×50=2500 pairs. Each subset of this product is a value of the chassis relation. Since there are 2^{2500} subsets, there are that many possible values to enumerate. We can now try splitting the scope and redefining the specification; see Diplay 16. Alloy returns a model of EmptyCond for the scope consisting of 25 hubRouters, 25 spokeRouters, 25 hubRouterInterfaces and 25 spokeRouterInterfaces in seconds! Note that the scope still contains 50 routers and 50 interfaces. But there are now "only" $2^{625} \times 2^{625} = 2^{1250}$ possible values of chassis relation, or a factor of 2^{1250} less. The scope splitting heuristic has been followed to structure the space of different routers and interfaces in the fault-tolerant VPN.

Minimizing Number Of Quantifiers In Formulas

Requirements containing quantifiers are transformed into Boolean form by instantiating quantified variables in all possible ways. The number of instantiations is the product of the number of instantiations of each quantified variable. The number of instantiations of each quantified variable is the size of the scope of that variable. In order to prevent the Boolean form from becoming excessively large, one can keep the number of quantified variables in a requirement as small as possible. For example, consider the definition of FirewallPolicyRequirements show in Display 17 in which only two explicit quantifiers appear per requirement. Display 18 shows a more compact definition in which five quantifiers appear in a single requirement.

```
pred internalSubnetAdvertisedToWan ()
      {some r:legacyRouter, x:physicalInterface, y:physicalInterface,
            s:internalSubnet, e:externalSubnet |
            x.chassis=r &&
            y.chassis=r &&
            x.network=s &&
            y.network=e &&
            x.routing=y.routing}
```

Display 14: Advertising internal subnet if several conditions met.

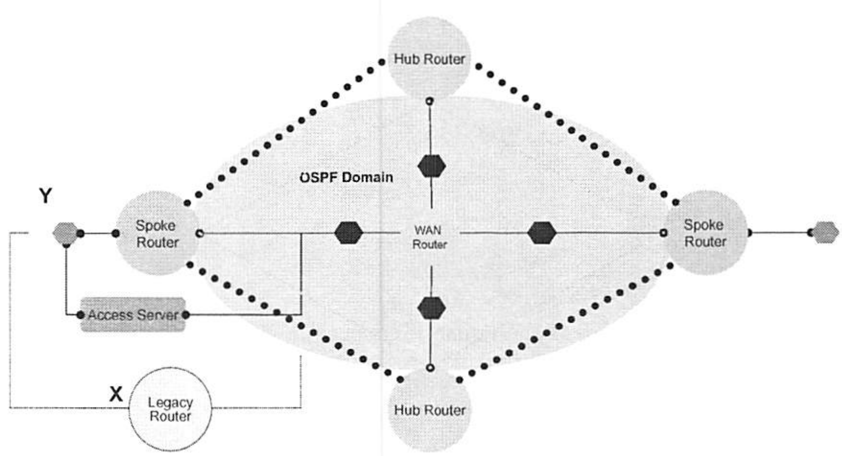


Figure 6: Advertisement of internal subnet Y into WAN by legacy router.

```
routing :  =
    {physicalInterface_0 -> ospfDomain_0,
     physicalInterface_1 -> ospfDomain_0,
     ..}
chassis :  =
    {physicalInterface_0 -> legacyRouter_0,
     physicalInterface_1 -> legacyRouter_0,
     ..}
network :  =
    {
     physicalInterface_0 -> externalSubnet_3,
     physicalInterface_1 -> internalSubnet_1,
     ..}
```

Display 15: Code for expanded specifications (FullVPNSpec ∧ internalSubnetAdvertisedToWan).

In both cases, the number of IPSec tunnels in the scope is 4 and the number of firewall policies 9. However, there is a large difference in the size of the Boolean formula produced (for the entire VPN specification). In the first case, the formula contains 216,026 clauses and 73,4262 literals, and the entire process (compilation to solution) took 2 minutes and 59 seconds. In the second case, the formula contains 601,721 clauses and 2,035,140 literals and the entire process took 8 minutes and 19 seconds.

Relationship To Previous Work

IETF's policy-based networking group [7] has similar objectives to ours. Its main contributions are vendor-neutral information models and *if-condition-then-action* rules called policies. Information models define types of objects, their attributes and possible values. These models, while important from a software development standpoint, are orthogonal to solving fundamental configuration management problems identified in this paper. These problems would remain even if we were to use all components from a single vendor.

Policy-based networking also does not enable any *declarative* representation of system logic such as Requirements 1-16. The *if-condition-then-action* rules are only procedural encodings of this logic. In effect, these rules have to do all the work of the Requirement Solver. This is a formidable undertaking. Furthermore,

```
sig hubRouter {}
sig spokeRouter {}
sig hubRouterInterface {chassis:hubRouter}
sig spokeRouterInterface {chassis:spokeRouter}
```

Display 16: Splitting the scope and redefining the specification.

```
pred FirewallPolicyRequirements ()
{(all t:ipsecTunnel | some p1:firewallPolicy |
      p1.protectedInterface = t.local &&
      p1.prot = ike &&
      p1.action = permit) &&
(all t:ipsecTunnel | some p1:firewallPolicy |
      p1.protectedInterface =t.remote &&
      p1.prot = ike &&
      p1.action = permit) &&
(all t:ipsecTunnel | some p1:firewallPolicy |
      p1.protectedInterface = t.local &&
      p1.prot = esp &&
      p1.action = permit) &&
(all t:ipsecTunnel | some p1:firewallPolicy |
      p1.protectedInterface = t.remote &&
      p1.prot = esp &&
      p1.action = permit)
(no disj p1,p2:firewallPolicy |
p1.protectedInterface=p2.protectedInterface &&
      p1.prot=p2.prot && !p1.action=p2.action)}
```

Display 17: Two explicit quantifiers per firewall requirement.

```
pred FirewallPolicyRequirements ()
{(all t:ipsecTunnel | some p1,p2,p3,p4:firewallPolicy |
      p1.protectedInterface = t.local &&
      p1.prot = ike &&
      p1.action = permit &&

      p2.protectedInterface = t.remote &&
      p2.prot = ike &&
      p2.action = permit &&

      p3.protectedInterface = t.local &&
      p3.prot = esp &&
      p3.action = permit &&

      p4.protectedInterface = t.remote &&
      p4.prot = esp &&
      p4.action = permit)&&
(no disj p1,p2:firewallPolicy |
p1.protectedInterface=p2.protectedInterface &&
      p1.prot=p2.prot && !p1.action=p2.action)}
```

Display 18: More compact version of Display 17.

verification with such procedural rules is impractical, and operations like configuration error diagnosis or fixing are not addressed. In our approach, the Requirement Solver remains unchanged. It is only the requirements or the scope that change. The Requirement Solver automatically adjusts to these changes and finds new configurations. Verification is another application of the Requirement Solver.

For the same reason, it is incorrect to assume that just the use of high-level languages like Perl and Python, often used in network management, can solve above fundamental configuration management problems. The hard part of reasoning from full first-order logic requirements still has to be programmed in these languages. It is this part that our approach automates.

Previous papers [8, 9] formalized a restricted version of the second section's requirements, in Prolog. Prolog is based in definite clauses, hence it is not possible to use it to reason with full first-order logic constraints. Examples of these are *"for every GRE tunnel there is an IPSec tunnel between associated physical interfaces that secures all GRE traffic"* and *"no two distinct interfaces on a router are on the same subnet."* The application of Prolog to system administration is thoroughly explored by Couch and Gilfix [10]. Related, widely used systems are Burgess' CFEngine [11] and Anderson's LCFG [12], but both have less expressive power than Prolog. These systems also perform robust application of configuration to components, a problem outside the scope of this paper. Recently, the need for specifying and reasoning with constraints on configurations has been amply expressed [13]. These constraints require the expressive power of full first-order logic, therefore our approach can address this need.

Summary, Conclusions & Future Directions

This paper introduces the notion of a Requirement Solver and shows how fundamental configuration management tasks can be naturally formalized using it. These tasks are configuration synthesis, requirement strengthening, component addition, configuration error diagnosis and configuration error fixing. The Solver has been inspired by, and is implemented in, the new logical system called Alloy. Alloy is based on the concept of model finding for full first-order logic in *finite* domains. Because of Alloy's use of highly efficient satisfiability solvers, there is renewed optimism for efficient reasoning in this logic, especially for configuration management. Traditional first-order logic theorem provers address the harder problem of reasoning in *infinite* domains. The Solver is illustrated in the context of a realistic fault-tolerant VPN with remote access, by working out four of above tasks. Approaches for writing efficient specifications are outlined.

Alloy's strength is efficiently sorting through complex, first-order logic constraints, provided scopes are small. On a modern PC, it requires several hours to find complete configurations for all components in an 8-site VPN: 8 spoke routers, 2 hub routers, 1 access server, 1 WAN router and associated interfaces, subnets, firewall policies, routing domains, GRE and IPSec tunnels. For context, the time taken by a human to reconcile Requirements 1-16 for all sites should be considered. Based on experiments of this paper, it is possible that Alloy be used from a traditional programming language to solve configuration management problems for networks of realistic scale and complexity. The heuristics of the "Efficient Requirements" section could be programmed in the programming language. Other approaches for scalability are tuning satisfiability solvers to the networking domain, improving Alloy compilers, and using divide-and-conquer approaches.

One open problem is selecting the least cost change from the current configuration as required for Configuration Error Fixing. A configuration management problem, not discussed in this paper, is migration planning: in what order to configure components so that mission-critical invariants are never violated. For example, suppose the routing protocol on all routers has to be changed from RIP to OSPF. If the only method of accessing routers to perform this change is inband, then reconfiguring the first router to which the management station is attached will effectively isolate it from all others. This is because routing protocols compute routes to routers, but since OSPF and RIP processes do not exchange information with each other, the first router will not be able to compute routes to others. The problem of the order in which to reconfigure components is fundamentally the problem of planning in artificial intelligence. The application of satisfiability solvers to this problem has been shown by Selman and Kautz [14].

Acknowledgements

I am grateful to Dr. Paul Anderson at Edinburgh, Professor Mark Burgess at University of Oslo, Professor Carla Gomes at Cornell, Professor Daniel Jackson at MIT, Dr. Gary Levin at Telcordia, Professor Sharad Malik at Princeton, and Professor Darko Marinov at University of Illinois, Urbana Champaign for very useful ideas and feedback.

Author Information

Sanjai Narain is a Senior Research Scientist in the Information Assurance and Security Department in Telcordia's Applied Research Area. His current research is on automated synthesis of secure, fault-tolerant distributed systems. This research is funded through DARPA, DISA and Department of Homeland Security. He has built security and network management systems for wireless, IP, VoIP, DSL, Dialup, ATM and SONET. The DSL loop qualification system he created was the basis for a successful Telcordia service. He won

a DARPA award for transferring technology to the Army's Future Combat Systems program. The DR. DIALUP system he created for reducing help-desk costs of Internet Service Providers was Telcordia's first product for the mass-market. Prior to joining Telcordia, he worked at RAND Corporation where he designed and implemented new discrete-event simulation techniques. His formal training is in programming languages and automated reasoning. He obtained a Ph.D. in Computer Science from University of California, Los Angeles, an M.S. in Computer Science from Syracuse University, and a B.Tech. in Electrical Engineering from Indian Institute of Technology, New Delhi. His email address is narain@research.telcordia.com .

References

[1] Oppenheimer, David, Archana Ganapathi, David A. Patterson, "Why Internet Services Fail and What Can Be Done About These?" *Proceedings of 4th Usenix Symposium on Internet Technolo gies and Systems (USITS '03)*, http://roc.cs.berkeley.edu/papers/usits03.pdf, 2003.

[2] Lampson, Butler, "Computer Security In the Real World," *Proceedings of Annual Computer Security Applications Conference*, http://research.microsoft.com/lampson/64-SecurityInRealWorld/Acrobat.pdf, 2000.

[3] Horn, Paul, Senior VP, IBM Research, *Autonomic Computing IBM's Perspective on the State of Information Technology*, http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.

[4] http://alloy.mit.edu/ .

[5] *Berkmin*, http://eigold.tripod.com/BerkMin.html .

[6] *Zchaff*, http://www.princeton.edu/~chaff/ .

[7] Moore, B., E. Ellesson, J. Strassner, A. Westerinen, "Policy Core Information Model – Version 1 Specification," *IETF RFC 3060*, http://www.ietf.org/rfc/rfc3060.txt, February, 2001.

[8] Narain, S., T. Cheng, B. Coan, V. Kaul, K. Parmeswaran, W. Stephens, "Building Autonomic Systems Via Configuration," *Proceedings of AMS Autonomic Computing Workshop*, Seattle, WA, http://www.argreenhouse.com/papers/narain/Autonomic.pdf, 2003.

[9] Qie, X., S. Narain, "Using Service Grammar to Diagnose Configuration Errors in BGP-4," *Proceedings of Usenix Systems Administrators Conference*, San Diego, CA, 2003. Also to appear in *Science of Computer Programming Journal*, in a special issue on Network Management.

[10] Couch, A., M. Gilfix, "It's Elementary, Dear Watson: Applying Logic Programming To Convergent System Management Processes," *Proceedings of Large Installations Systems Administration Conference (LISA)*, http://www.eecs.tufts.edu/~mgilfix/publications/prolog-lisa99.pdf, 1999.

[11] Burgess, M., "A Site Configuration Engine," *USENIX Computing systems*, Vol. 8, Num. 3, http://www.iu.hio.no/~mark/papers/paper1.pdf, 1995.

[12] Anderson, P., A. Scobie, "LCFG – The Next Generation," *Proceedings of UKUUG LISA/Winter Conference*, http://www.lcfg.org/doc/ukuug2002.pdf, 2002.

[13] Configuration Workshop, Large Installations Systems Administration Conference (LISA), http://homepages.informatics.ed.ac.uk/group/lssconf/config2004/index.html, 2004.

[14] Selman, B., H. Kautz, "Planning As Satisfiability," *Proceedings of ECAI-92*, http://www.cs.cornell.edu/selman/papers/pdf/92.ecai.satplan.pdf.

Visualizing NetFlows for Security at Line Speed: The SIFT Tool Suite

William Yurcik – National Center for Supercomputing Applications (NCSA)

ABSTRACT

The first step in improving Internet security is measurement – security events must be made visible. The irony in making this happen is that there is no lack of security measurement data, in fact, quite the opposite. However, making security manifest faces a major challenge: the large volume and multi-dimensional nature of security data typically obscures valuable security events. NCSA has developed a suite of tools that solves this problem and is making this software available to the Internet community.

We present two visualization tools,[1] (1) NVisionIP and (2) VisFlowConnect-IP. Both of these tools have been developed based on system administrator requirements, their design peer-reviewed in security research forums, and usability testing is in process. These tools both present large volume complex data transparently to system administrators in simple intuitive visual interfaces that support human cognitive processes. NVisionIP visually represents the state of all IP addresses on large networks on a single screen window (we use a Class B address space as the default) with capabilities to filter and drill-down to subnets and individual machines for details-on-demand. VisFlowConnect-IP visually represents flows between internal network IP hosts and the Internet showing who is connecting with whom with capabilities to filter and drill-down to subnets and individual machines for details-on-demand. NVisionIP and VisFlowConnect-IP can be used individually or in unison for correlating events. This work is distinguished from others in that these are the first Internet security visualization tools to be freely available on the Internet and deployed in large production environments.

Introduction

Organizations use computer network infrastructures that hold a vast amount of information for system administrators and security engineers. There are typical logs common to most computer networks, but the systems are often large and dynamic, making it difficult to extract knowledge from the sea of information. Individually, each system log can be massive, causing operator overload. When overload occurs, security events can slide by unnoticed. Overload can also cause operators to disregard alarms due to high false positive rates. Even in homogeneous infrastructures, solutions from a single vendor fail to scale to medium or large networks. However, the problem is compounded because most organizations have network infrastructures from multiple vendors.

We have developed Security Incident Fusion Tools (SIFT) [9], an integrated suite of tools for evaluating the security of an entire computer network on a single screen. We address the need to discover security incidents that currently go undetected by security operations systems. Specifically two SIFT tools, (1) NVisionIP and (2) VisFlowConnect-IP, leverage human visual cognitive abilities to process log data into knowledge for situational awareness of network security. It is estimated that human beings can visually process a screen of information at 150 Mbits per second [10], with the ability to discriminate relatively minor shifts in color, shape, and motion. By presenting network data visually, it can be scanned quickly, patterns in complex data rise to the surface, and inferences become intuitive. Once a security professional becomes familiar with the normal appearance of the network being monitored, it is much easier to spot attacks including new so-called "zero-day attacks." The tools are designed to give security engineers situational awareness of an entire network in order to help them determine when a network is under attack, what is being attacked, and what form the attack is taking.

The remainder of the paper is organized as follows: The next section discusses NetFlows data management and introduces the first tool in the SIFT suite: CANINE. Subsequently we present the two SIFT visualization tools – NVisionIP and VisFlowConnect-IPand close with a summary and on-going future work.

Data Management

NetFlows Source Data

While this paper focuses on visualization, we would be remiss if we did not address data management since it is arguably the greatest obstacle in realizing any scalable visualization system. We address the challenge of processing high-bandwidth data streams by instrumenting networks with distributed NetFlows

[1]Funded in part by grants from the Office of Naval Research (ONR) under the auspices of the Technology Research, Education, and Commercialization Center (TRECC) and the National Center for Advanced Secure Systems Research (NCASSR) both established at NCSA/University of Illinois.

sensors and then combining this sensor data into a unified format. While in the recent past NetFlows were solely router-based, PC-based NetFlow sensors (Argus) make this a feasible solution for most organizations. The first tool in our suite is a NetFlows converter/anonymizer called CANINE which can handle different NetFlows formats so independent implementations can be interoperable with SIFT visualization tools. NetFlow logs have proven to be the appropriate granularity to process heavily loaded networks and high bandwidth connections (Gb/s) in near-real-time (five minute monitoring windows).

A *network flow* is defined as a sequence of packets that are transferred between two endpoints within a certain time interval. The endpoints are identified at the network layer by IP addresses and at the transport layer by port numbers. In addition to data format differences, there are other interoperability problems in practical NetFlows implementations:

- Cisco NetFlows are defined as *unidirectional* and generated through intelligent flow cache management, which contains a set of specialized algorithms [4].
- Argus NetFlows are defined as *bidirectional* containing two distinct sub-flows, one in each direction [2].
- Cisco and Argus NetFlow formats have different fields (e.g., flags etc.) [3, 5].

For a more detailed comparison between different NetFlows formats see [15].

CANINE

With the increased use of NetFlows for security monitoring and the fact that NetFlows come in different and incompatible formats, we have developed CANINE (Converter and ANonymizer for Investigating Netflow Events) [7, 8] which can be downloaded from http://security.ncsa.uiuc.edu/distribution/CanineDownLoad.html. CANINE allows tools designed for a specific type of NetFlows to be interoperable with any NetFlow format. CANINE consists of the two main modules: (1) the CANINE GUI and (2) the conversion/anonymization engines. For the purposes of this paper we will only discuss the conversion engine (for information about the anonymization engine see [7, 8]). The CANINE GUI accepts user input to identify the NetFlow file for conversion, sends the request to the processing engine which performs the conversion to the newly specified output file, and lastly summarizes the results of the performed actions in a popup window. At present CANINE supports conversion to/from Cisco version 5/7, Argus, NFdump, and our own NCSA internal NetFlows format. Future formats to be included in CANINE include Cisco version 9 and the future IETF IPFIX standard.

Network Instrumentation

With the development of high-speed network infrastructure has also come the need for high-speed security – security at line speed – for current 2005 networks this is 4 GB/s at the edge and higher within the core [15]. Unfortunately, high network bandwidths present special problems for security monitoring.

The first challenge is the streaming nature of security sensors. It is important to note that security sensors generate streaming data and not batch log files. Since streaming analysis is an open research question, security systems typically create batch log files by collecting streaming data over defined time periods. However, depending on the network size and traffic volume these log files can become large and difficult to handle. Tuning is required to determine the best time period of analysis to match the preferred log size to the network size and traffic volume. Creating logs over longer time intervals may risk losing NetFlows records upon high transmission rates from overflow or blocking.

The second challenge is observation point. Security cannot be measured where it is not observed thus sensors need to be placed to cover the entire network space. Typical deployment for NetFlows includes the border router for Internet traffic and Argus sensors for internal network observation. There are blind spots from VLANs and switched networks which do not leave IP (network layer) traces – future sensors based on S-Flows are developing to address this gap.

The third challenge is CPU speed to generate and process NetFlows at line speed. As routers have increased speed, monitoring techniques have shifted to sampling NetFlows. While sampling is statistically sufficient for network planning, it is not a good idea for security analysis. NetFlow records are created by sampling packets (not flows), letting the majority of the packets go unnoticed, which may lead to missing important security events. A possible justification for sampling is that an attack may be high traffic volume, at least part of which may be captured with high probability (such as high-volume denial-of-service attack or indiscriminate scanning by propagating worms and viruses). A preferred approach we recommend for security at line speed is the parallel processing NetFlows in a distributed manner. Instead of instrumenting only the high-speed border router that may only be able generate sampled NetFlows, instead instrument all the routers feeding into the border router. This technique effectively relieves the load on each flow collector so that it will not be over subscribed. The drawback is that multiple flow collectors are required and NetFlows records from different routers must be merged to eliminate duplicate flows (the same flow that passes through multiple routers).

NetFlows Visualization Tools

Design By Requirements

We firmly believe that the first step to improve Internet security is by measurement. Measurement

allows one to accurately assess the degree of the problem at a specified time and then further measurements track whether solutions are having the desired effect. However, not all measurements are equal, users have a mental model based on experience and tools should be designed to enhance and augment these mental models for the most effective results [16, 17, 18].

For this work, we did two important things often neglected from security tool design: (1) taking time to work with security engineers in their operational production environment in order to learn their mental models and thus tool requirements and (2) the capability to design new visualization models from scratch to meet these requirements without having to incorporate legacy constructs. The results have been very satisfying in that most security engineers who view our visualization tools for the first time immediately begin inferring hypotheses based on the content displayed.

To briefly summarize the major findings from our requirements analysis there are two primary findings. First, security engineers need to answer questions such as these posed by upper management: What is the state of the network? Is the network being attacked? How is the network being attacked? Who is attacking the network? While these may appear to be basic questions, the answers are not immediately available using current security tools and when available after much analysis the answers are complex. Visualization provides a rich representation to help answer these questions concisely.

Second, security engineers have mental models based on their experience with the network infrastructure, knowledge of people within the organization, and security expertise learned over many years. While most tenets of information visualization design are useful in designing within our specific security domain, we did find that leveraging the mental model of security engineers caused us to break some of these consensus rules (after much consternation). Instances when the security engineer mental model overrides information visualization design best practices are highlighted in our discussion of each of our visualization tools.

NVisionIP

Our first and most mature security visualization tool is NVisionIP [1, 6] which we designed to answer the question: What is the state of the network?

Figure 1 shows the Galaxy view of NVisionIP which can be downloaded from http://security.ncsa.uiuc.edu/distribution/NVisionIPDownLoad.html. The Galaxy view represents an entire Class B IP address space (in this single window!) as a matrix with subnets along the horizontal axis and hosts along the vertical axis. Each IP address is represented as a dot (actually four pixels) and the state of each IP address is represented with color or shape as determined by the user in the color and shape legend. Two magnification options are available to see the IP addresses: linear and fisheye.

NVisionIP allows the security engineer is to load one (or multiple) NetFlow files and perform visual queries. NVisionIP has taken all the possible NetFlow database query combinations and hard coded them into the tool as drop-down and point-and-click commands. A user would typically start with primary queries such as how bytes per IP address or how many connections per IP address. A filter then allows the user to select secondary queries to view only source or destination traffic, different protocol (IP, UDP), and different ports (destination or source ports, specific ports or collections of ports) or any combination thereof.

At the Galaxy view, NVisionIP can identify large or small levels of traffic as measured in bytes (based on expectations for the class of machine – laptop or server). This may indicate malware is being served to/from a machine or the machine is involved in a denial-of-service event. Worm and virus scans as indicated by number of connections can also be easily detected based on variance from expected levels.

There are aspects of the Galaxy view design that are contrary to information visualization best practices: the IP address space is laid out logically in matrix space without organizing IP addresses into known classes or enlarging the part of the IP address space with more activity (thus patches of white space or inactive IP address space appears). This design was intentional to retain security engineer knowledge of the IP address space based on logical numbering for subnets/hosts and mental mapping between logical addresses (e.g., cluster compute nodes with contiguous IP addresses) and physical locations (IP subnets are usually physically located in the same area such as a building floor etc.). The white space of inactive IP addresses actually has other advantages and is not wasted space – any traffic activity shown there is anomalous (unallocated address space that should have no legitimate traffic).

While an overall view is important, it is of limited use without the ability to drill down to find more detailed information when something interesting is identified. Figure 2 shows the drill-down levels of NVisionIP which are activated with a mouse click and a drag over a region of interest. These levels are the (1) Small Multiple View and (2) Machine View.

The Small Multiple View allows the user to quickly scan and compare traffic activity across subnets on many machines simultaneously. Each machine is a box with two sets of histograms, an upper set of histograms representing traffic on well-known ports and a lower set of histograms representing traffic on ports over 1024. The well-known ports are color-coded in a user legend. The ports over 1024 are ordered from most active to least active (top N ports). Note that no numbers are shown in the small multiple view, this view is designed for the user to identify activity of interest and then drill-down for raw data details on-demand.

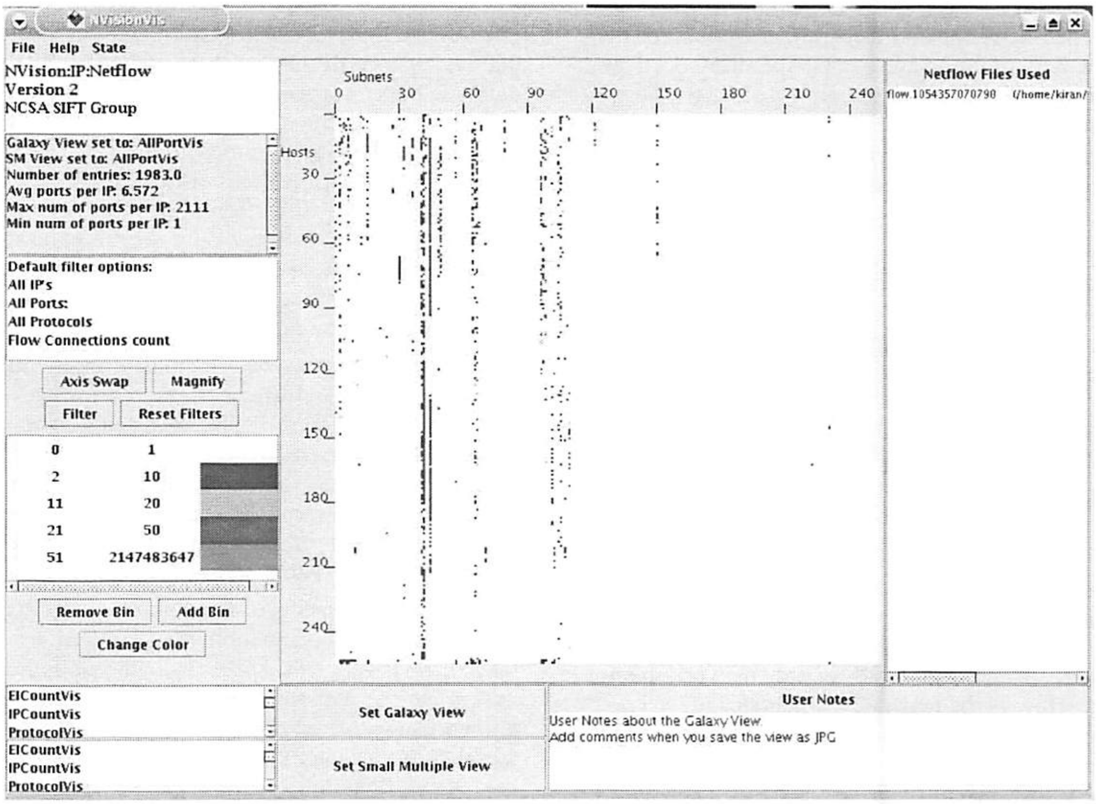


Figure 1: NVisionIP Galaxy view of an entire Class B IP address space.

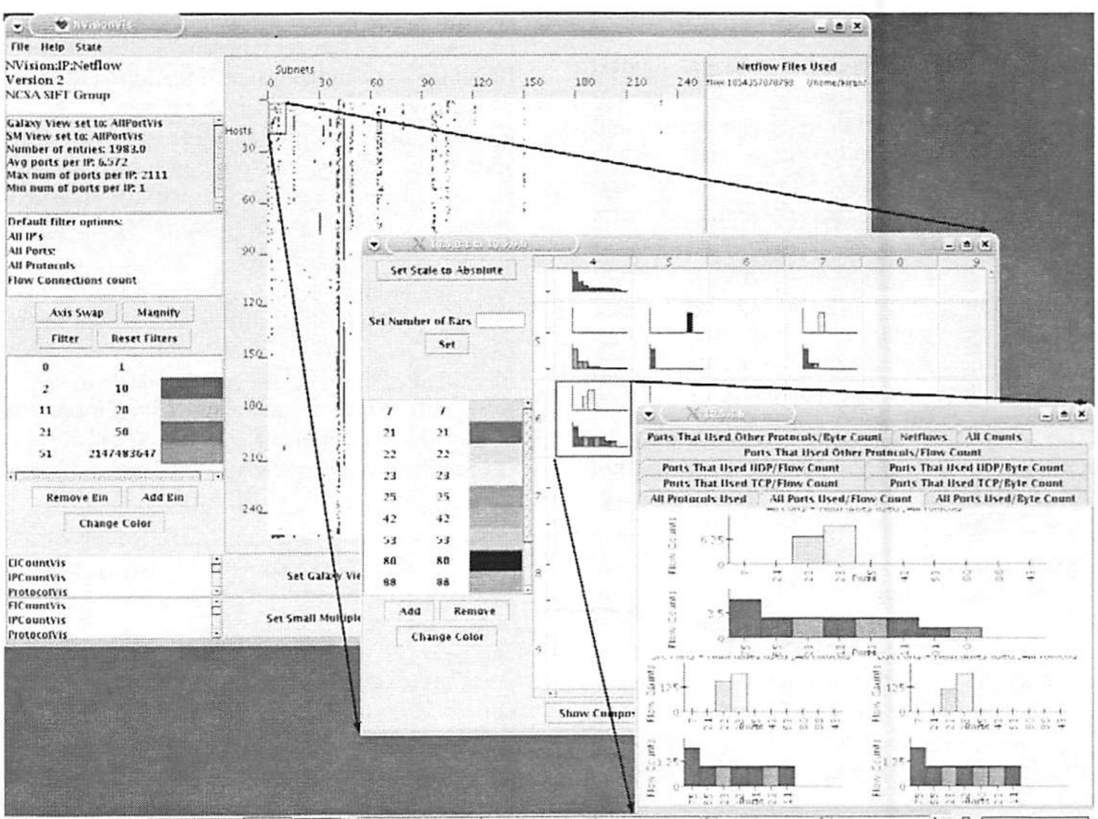


Figure 2: The three levels of NVisionIP (top to bottom): (1) Machine view, (2) Small multiple view, and (3) Galaxy view.

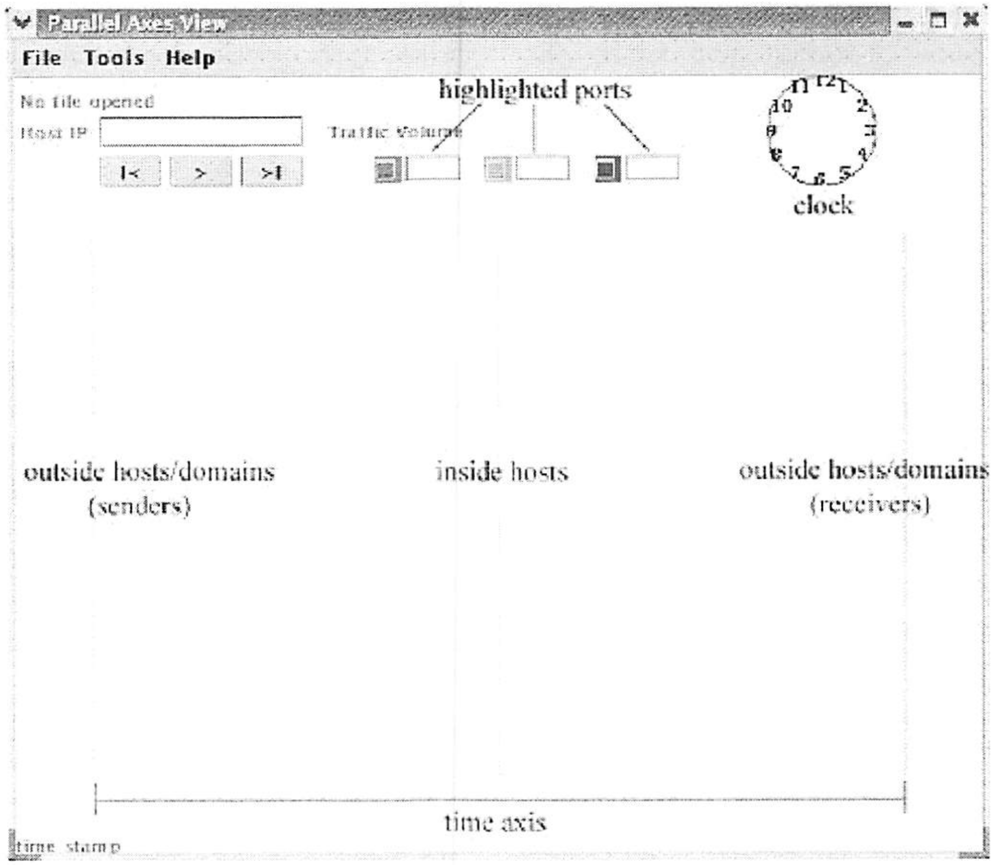


Figure 3: VisFlowConnect-IP: Main view.

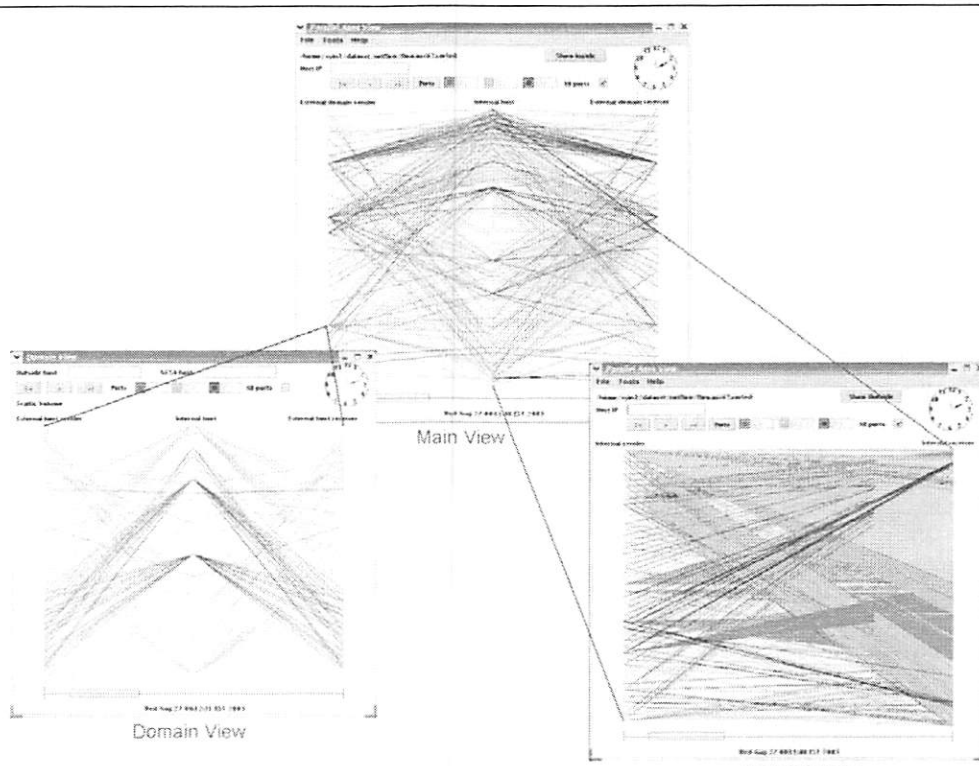


Figure 4: Drill-down layers of VisFlowConnect-IP: left Domain View and Bright Internal View.

At the Small Multiple View, NVisionIP has been used to quickly identify anomalous services that violate security policy such as unauthorized services (web or mail server) and exposed services that need to be patched or otherwise protected due to vulnerabilities.

If a user sees a machine with interesting traffic activity in the Small Multiple View, they may click on that block to drill-down to the Machine View. The Machine View organizes all the data from a particular machine in multiple tabs – each tab showing aggregate activity in an upper histogram and source/destination directional flows in two lower histograms. Note in the Machine View each histogram is fully labeled with port numbers and traffic level. At the lowest level, the raw NetFlows data for that machine is available for inspection in this Machine View as a tab. At this lowest level Machine View, details of most security events can be revealed.

The overall effect of using NVisionIP (with its interactive drill-down levels) is that relationships between aggregate network activity and individual machine activity can be more easily discovered and comprehended by human operators – providing situational awareness of network system state. Future work is progressing on optimizing Galaxy View animation to show IP address state changes over time as well as a difference view to visually compare current network traffic versus benchmark network traffic.

VisFlowConnect-IP

The second security visualization tool in the SIFT tool suite focuses on answering the question: Who is connecting to whom on the network? This basic question has been attempted in the past with topology-based diagrams based on network infrastructure, however, these results were either (1) not providing information relevant to real-time traffic or (2) not scalable since representing network traffic over time produces obscured lines in relatively short order. We solve both of these problems with VisFlowConnect-IP [12, 13, 14] which is available at http://security.ncsa.uiuc.edu/distribution/VisFlowConnectDownLoad.html.

VisFlowConnect-IP is a security visualization tool based on the parallel axes concept drawn from data mining. It is a complementary tool to NVisionIP since it visualizes the same NetFlows source data – the design similarities will become apparent in the following description. VisFlowConnect-IP allows a user to visually assess the connectivity of large and complex networks (in a single window!) by providing a main view of the network with filter and drill-down views that provide more details on-demand. The three views of VisFlowConnect-IP are: (1) Main, (2) Domain, and (3) Internal. The Main View is shown in Figure 3 with the Domain and Internal views shown in Figure 4.

The VisFlowConnect-IP Main View utilizes the parallel axis view with the left-most and right-most vertical axes representing the external domains and the center vertical axis representing host IP addresses within the internal edge network domain (See Figure 3). Lines connecting external domains and internal hosts represent directional data flows, with line darkness being proportional to the logarithm of the volume of data transferred. VisFlowConnect-IP can filter/highlight flows to certain hosts or traffic on specific ports and protocols using a filter drop-down menu and selection boxes on the main view. Ports indicated in the selection boxes are represented in different colors within the network traffic or may be isolated from network traffic for focused analysis. The overall effect is visualization of traffic into-an-edge-network-from-the-Internet and traffic out-from-an-edge-network-to-the-Internet.

Figure 4 shows the two drill-down views within VisFlowConnect-IP. While we would have liked to represent each individual external host IP address connecting into the internal edge network symmetrically on both the left-most and right-most axes, this is not possible due to scalability. Preliminary measurements of NCSA's network showed over 100,000 different IP addresses commonly appeared in the NetFlow files we wished to visualize and this is too many for the vertical line pixel space of a single window without scrolling. Instead we implemented a drill-down Domain View which is invoked by the user clicking on a drop-down menu while having an external domain highlighted on the vertical external domain axis. The resulting Domain View is a mirror image of the Main View except it only shows traffic within the highlighted external network domain to/from the internal edge network. This has turned out to be very valuable since typically hackers "own" entire subnets or even "own" entire network domains so it is common to see malicious activity captured within a Domain View.

Figure 4 also shows the drill-down Internal View which is invoked as a toggle button on the Main View. While monitoring for external Internet hacker activity is sexy, we have found this Internal View very useful since it shows only traffic that both sources and sinks within the internal edge network. There are only two vertical lines in this view, internal edge network IP addresses are ordered symmetrically in a mirror image on the left-most and right-most axes (no middle axis). This Internal View has helped security engineers determine important security events like the initial source of a worm infection which infiltrated the edge network from the inside, and the insider attacks from those misusing privileged access.

The VisFlowConnect-IP Main View has a time axis at the bottom which is used to solve the scalability problem we referred to as the major challenge for this tool. The user loads a NetFlow file for visualization and then may select multiple filters to determine how this traffic is to be represented in animation including intensity, byte size, and a sliding time window. The sliding time window provides scalability by only representing traffic within the window and ignoring traffic outside the window. Thus the sliding time window can be adjusted to any size network and any

traffic volume – the general rule for clear viewing is the more traffic the smaller the sliding time window. The window size itself is represented to the user by a red box (where the length of the red box is proportional to window size) that travels along the time axis as the traffic is animated (as shown in the Domain and Internal Views within Figure 4).

VisFlowConnect-IP has also implemented a filter language using real expressions that is beyond the scope of this paper [12]. With this filter language capability, VisFlowConnect-IP can create mechanisms for storing/retrieving filter profiles. These profiles can store customized filters that remove "uninteresting" information from view-thus leaving only the more security relevant data to be displayed.

Summary

Visualization is the future of security monitoring and NetFlows are the source data for high-speed networks. In this paper we marry security visualization with NetFlows by presenting the SIFT suite of tools along with accompanying techniques for security at line speed. The goal is to enable security engineers to go beyond binary/text command line log file analysis toward real-time network security situational awareness. A growing community of researchers has formed on security visualization, see [11] for more information.

The three specific tools of the SIFT suite presented in this paper (CANINE, NVisionIP, and VisFlowConnect-IP) are available for download at the URLs provided in the text. We are currently conducting usability tests with human subjects to quantify the utility of these tools and preliminary results from these tests are very promising. We intend to go open source with these tools after the software is stable, at present we are still developing the software with new versions posted on the corresponding webpages. We enthusiastically invite feedback from users about the use of these tools.

Author Biography

William (Bill) Yurcik is currently Manager, Security R&D and Senior Systems Security Engineer at NCSA. Prior to this he was Head of Security Operations at NCSA, so he has both a theoretical and practical background in computer network security. Prior to joining NCSA he has 12 years of professional experience as a Network Engineer for large networks (Naval Research Laboratory, NASA, Verizon, and MITRE). He is a graduate of Johns Hopkins University (MS Electrical Engineering 1990, MS Computer Science 1987), the University of Maryland (BS Electrical Engineering 1984), and is Ph.D. ABD from the University of Pittsburgh (1994-99). Bill can be reached at byurcik@ncsa.uiuc.edu .

References

[1] Bearavolu, Ratna, Kiran Lakkaraju, and William Yurcik, "NVisionIP: An Animated State Analysis Tool for Visualizing NetFlows," *FLOCON*, 2005.

[2] Bullard, Carter, *Argus, the network Audit Record Generation and Utilization System*, http://www.qosient.com/argus/ , accessed 26 September, 2005.

[3] Bullard, Carter, *Argus Record Format*, http://www.qosient.com/argus/argus.5.htm/ , accessed 26 September, 2005.

[4] Cisco Systems, *Cisco NetFlow Services and Applications White Paper*, http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm , accessed 26 September, 2005.

[5] Cisco Systems, *NetFlow Overview Presentation*, http://www.cisco.com/application/vnd.mspowerpoint/en/us/guest/tech/tk362/c1482/ccmigration_09186a0080182b50.ppt , accessed 26 September, 2005.

[6] Lakkaraju, Kiran, William Yurcik, Adam J. Lee, Ratna Bearavolu, Yifan Li, and Xiaoxin Yin, "NVisionIP: NetFlow Visualizations of System State for Security Situational Awareness," *CCS Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC)*, 2004.

[7] Li, Yifan, Adam Slagell, Katherine Luo, and William Yurcik, "CANINE: A Combined Converter and Anonymizer Tool for Processing NetFlows for Security," *International Conference on Telecommunication Systems – Modeling and Analysis (ICTSM)*, 2005.

[8] Luo, Katherine, Yifan Li, Adam Slagell, and William Yurcik, "CANINE: A NetFlows Converter/Anonymizer Tool for Format Interoperability and Secure Sharing," *FLOCON*, 2005.

[9] *SIFT Project Webpage*, http://www.ncassr.org/projects/sift/ , accessed 26 September, 2005.

[10] Tufte, Edward, *A One-Day Course: Presenting Data and Information*, Madision WI, (http://www.edwardtufte.com/tufte/courses , accessed 26 September, 2005), August, 2005.

[11] *VizSEC Community Homepage*, http://www.ncassr.org/projects/sift/vizsec/ , accessed 26 September, 2005.

[12] Yin, Xiaoxin, William Yurcik, and Adam Slagell, "VisFlowConnect-IP: An Animated Link Analysis Tool for Visualizing NetFlows," *FLOCON*, 2005.

[13] Yin, Xiaoxin, William Yurcik, and Adam Slagell, "The Design of VisFlowConnect-IP: a Link Analysis System for IP Security Situational Awareness," *Third IEEE International Workshop on Information Assurance (IWIA)*, 2005.

[14] Yin, Xiaoxin, William Yurcik, Michael Treaster, Yifan Li, and Kiran Lakkaraju, "VisFlowConnect: NetFlow Visualizations of Link Relationships for Security Situational Awareness," *CCS Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC)*, 2004.

[15] Yurcik, William and Yifan Li, "Case Study: Instrumenting a Network for NetFlow Security Visualization Tools," *21st Annual Computer Security Applications Conference (ACSAC)*, 2005.

[16] Yurcik, William, Kiran Lakkaraju, James Barlow, and Jeff Rosendale, "A Prototype Tool for Visual Data Mining of Network Traffic for Intrusion Detection," *Workshop on Data Mining for Computer Security (DMSEC)*, 2003.

[17] Yurcik, William, James Barlow, and Jeff Rosendale, "Maintaining Perspective on Who Is The Enemy in the Security Systems Administration of Computer Networks," *ACM CHI Workshop on System Administrators Are Users, Too: Designing Workspaces for Managing Internet-Scale Systems*, 2003.

[18] Yurcik, William, James Barlow, Kiran Lakkaraju, and Mike Haberman, "Two Visual Computer Network Security Monitoring Tools Incorporating Operator Interface Requirements," *ACM CHI Workshop on Human-Computer Interaction and Security Systems (HCISEC)*, 2003.

Interactive Traffic Analysis and Visualization with Wisconsin Netpy

Cristian Estan and Garret Magin – University of Wisconsin-Madison

ABSTRACT

Monitoring traffic on important links allows network administrators to get insights into how their networks are used or misused. Traffic analysis based on NetFlow records or packet header traces can reveal floods, aggressive worms, large (unauthorized) servers, spam relays, and many other phenomena of interest. Existing tools can plot time series of pre-defined traffic aggregates, or perform (hierarchical) "heavy hitter" analysis of the traffic.

Wisconsin Netpy is a software package that goes beyond the capabilities of other existing tools through its support for interactive analysis and novel powerful visualization of the traffic data. Adaptive sampling of flow records ensures that the performance is good enough for interactive use, while the results of the analyses stay close to the results based on exact data. Among the salient features of the package are: hierarchical analyses of source addresses, destination addresses, or applications within aggregates identified by user-defined filters; time series plots that separate the traffic into categories specified with ACL-like syntax at run time; interactive drill-down into analyses of components of the traffic mix; "heatmap" visualization of traffic that describes how two "dimensions" of the traffic relate to each other (e.g., which sources send to which destinations, or which sources use which service, etc.).

Introduction

The unrestricted packet communication supported by the Internet offers immense flexibility to the endhosts in how they use the network. This flexibility has enabled the deployment of new applications such as the web long after the IP protocol has been standardized and has contributed significantly to the success of the Internet. On the other hand, network operators want to monitor and to some extent control how their networks are used. Firewalls, network address translation, and traffic shaping boxes offer a degree of control that helps keep networks manageable. But even within the constraints of the policies implemented through these devices, the network traffic is very variable and traffic monitoring is necessary.

An analysis of network traffic can reveal important usage trends such as the application mix and the identity of the heaviest traffic sources or destinations. Sometimes these analyses can reveal misuses of the network: compromised desktop computers turned into spam relays, remote computers scanning the network for vulnerabilities, network floods directed against a single victim, or caused by a worm trying to spread aggressively. It is often the case that the analysis is urgent because it is carried out to explain a degradation in network service. It is also often the case that the network administrator does not know in advance which ports or IP addresses to focus on and he goes through an iterative process before being able to find convincing evidence for the cause of the problem. Fortunately there are many traffic analysis and visualization tools to assist the network administrator in the task of exploring and understanding the traffic carried by their network. Wisconsin Netpy is a new and powerful addition to this large family.

Related Work

Tobias Oetiker's MRTG [12] is an early traffic visualization tool widely used by network administrators to track the volume of IP traffic based on SNMP counters provided by routers and switches. His RRD-tool [12] provides support for visualizing time series plots of arbitrary data and it is actually used by all traffic visualization applications discussed in this section. Jeff Allen's Cricket [1] takes MRTG's idea one step further by allowing the user to track a large number of variables (say the traffic of various links in the network) using a scalable config tree. These tools offer visual information only about the volume of the traffic, but nothing on the composition of the traffic mix.

The NetFlow flow records generated by routers are an information source much richer than the SNMP counters. Toolkits such as OSU flow-tools [7] and SiLK [8] have tools for manipulating and analyzing NetFlow data. Typical analyses allow finding the top sources and destinations of traffic. cflowd [2] is a related package that also supports various types of traffic matrices. Dave Plonka's FlowScan [13] and Cristian Estan's AutoFocus [5] also provide time series of various predefined categories of traffic represented within the traffic mix. The supercomputing community is also interested in IP traffic visualization, and its members have written tools such as"the spinning cube of potential doom" [11] and NVisionIP [10].

AutoFocus and Ryo Kaizaki's aguri [9] employ a novel type of analysis related to top k reports called

hierarchical heavy hitters or traffic cluster analysis [6, 3]. This type of analysis is used extensively by Netpy.

Network administrators are often interested in the largest sources or destinations as measured in bytes, packets or flows. One important observation is that while existing tools give the exact traffic for these heavy hitters, the user can often accept small errors. In fact such small errors are already present if one uses sampled NetFlow. Netpy exploits this observation by sampling flow records to speed up traffic analysis and to reduce disk usage. For measuring the traffic in bytes or packets we use the "smart sampling" of flow records introduced by Duffield, et al., [4].

Traffic Analysis With Netpy

The user can direct Netpy to perform traffic analyses through a graphical user interface or through a console that supports interactive queries as well as scripts. All analyses use an intermediary database of flow records (see "The Structure of netpy" for more details about Netpy's structure). But what kind of traffic analyses can one perform with Netpy? That's the question we answer through the rest of this section.

Time Series Plots With User Defined Categories

Time series plots are an easy to read visual representation of the traffic. Existing tools such as FlowScan and AutoFocus allow the network administrator to define various traffic categories based on port numbers or network prefixes and have them plotted with separate colors. This way the plots reveal information about the cause of various spikes. Furthermore with separate plots measuring the traffic in bytes, packets, (and flows in a future version) the user will be able to detect not only large floods, but also scans that generate many flows, but not many bytes.

Netpy also supports these types of time series plots. The user can specify the categories using an ACL-like syntax: each rule specifies a source and destination prefix, protocol number and source and destination port range; flows are mapped to the category associated with the first rule they match. For FlowScan and AutoFocus the user needs to specify the categories of interest before the NetFlow data is "imported," whereas with Netpy the user specifies the categories at run time and it is quicker to recompute the plot after the user changes the ACL rules defining the categories because the analysis relies on the database, not on the large NetFlow files.

The Scope of Traffic Analysis

For time series plots, and all the other analyses, one needs to define which NetFlow records constitute the input to the analysis. Existing toolkits often allow the network administrator to configure separate traffic reports for separate links. Netpy separates NetFlow data into different links as data is imported into the database. When running an analysis, the user specifies which links' traffic to work with. The user also specifies the time interval of interest to the analysis.

The user can also specify a filter to apply to the data matching the previous two criteria. The filter consists of one or more rules similar to router ACLs (each rule specifies a source address prefix, destination address prefix, source port range, destination port range, and protocol number) and flow records that don't match any of the rules in the filter are not considered in the analysis. The GUI's interactive drill-

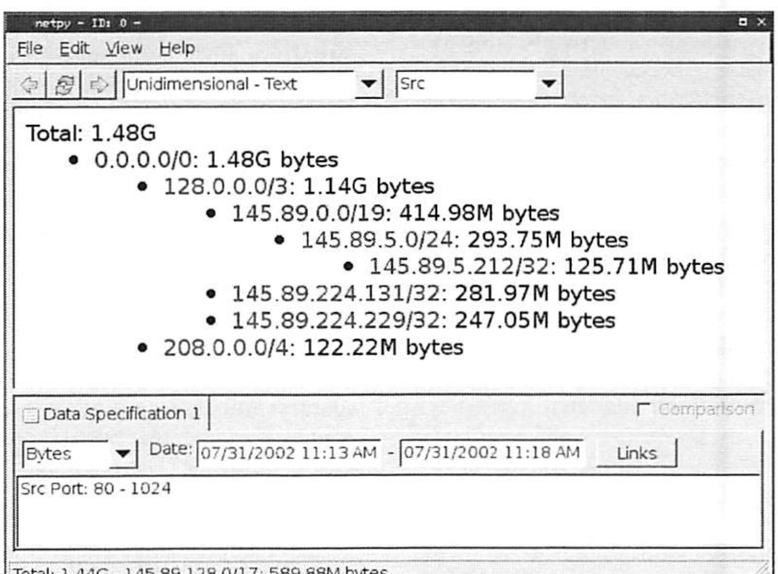


Figure 1: Hierarchical heavy hitter analysis on the sources of the traffic. Indentation is used to highlight prefixes including each other. The analysis finds the appropriate granularity based on the current traffic: 208.0.0.0/4 is a sixteenth of the address space and 145.89.5.212/32 is a single IP (all addresses are anonymized), but they are both reported because their traffic is above the threshold.

down feature works by setting the filter to select only the traffic of interest.

Hierarchical Heavy Hitters

The network administrator cannot always know in advance what port numbers or IP prefixes will dominate the traffic, so forcing her to specify in advance the ACL rules defining the categories doesn't always work. This is especially true after one drills down into a small, unfamiliar portion of the traffic mix. A traditional solution to this problem is to use "top K reports": one computes the traffic of each source address and reports the top K (say top 20). A related solution is the "heavy hitter report" which reports all sources whose traffic is above a given threshold in the data analyzed (say more than 1% of the total traffic). A problem with both these solutions is that they tell us nothing about sources that send little traffic: if for example we have a prefix with many small sources that nevertheless add up to a large portion of the traffic (a large modem pool), we would want to find out about their behavior. Netpy relies on the "hierarchical heavy hitter" algorithm that finds not just individual addresses, but also prefixes whose traffic is above a certain threshold specified as a percentage of the traffic being analyzed. This algorithm has the property that it not only identifies the prefixes generating significant traffic, but it also automatically finds the right prefix lengths to use when describing various portions of the traffic.

The hierarchical heavy hitter algorithm works as follows: first it reports all individual IP addresses whose traffic is above the threshold, next it aggregates the *remaining* traffic at the /31 level and reports any prefixes that are above the threshold, next it aggregates the remaining traffic at the /30 level and so on until it reaches the root of the IP address hierarchy. The criterion for reporting more general prefixes is that the difference between their traffic and the traffic of more specific prefixes already reported is more than the threshold. However, when Netpy reports such a prefix, it reports its total traffic not the difference between its traffic and that of more specific prefixes. Figure 1 shows the result of a hierarchical heavy hitter analysis on the source addresses in the traffic mix using a threshold of 5%. Through the threshold the user can control the level of detail: with a lower threshold, more prefixes are reported, with a higher threshold the user gets a coarser view.

The same type of hierarchical heavy hitter approach applies to destination IP addresses too. The approach actually generalizes to any hierarchy we can define on one or more of the packet header fields present in the flow record. To capture information about the applications in use, Netpy defines the following hierarchy: the first level below the root divides the flows by protocol, the second level divides the flows by source port into flows originating from low ports (0 to 1023) usually used by servers and high ports (1024 to 65535) usually used by clients, the third level divides the flows by actual source port value and the fourth level divides them by source and destination port value. The analysis will pick the granularity of the results based on the actual traffic. For example if there is a large TCP connection (e.g., a huge backup), the amount of traffic between its source port and destination port will be reported. If there is a source port used by many small connections (e.g., web traffic on port 80), the total traffic coming from port 80 will be reported. If there is no dominant source port, but the source ports used are in the high port range (e.g., traffic coming from a network of typical desktop computers). An example of this type of analysis is shown in Figure 2.

Bidimensional Analysis

The analyses looking at simple hierarchies such as the ones above can tell you that TCP port 80 and UDP port 53 generate a lot of traffic, and they can also tell you that servers A and B generate a lot of traffic,

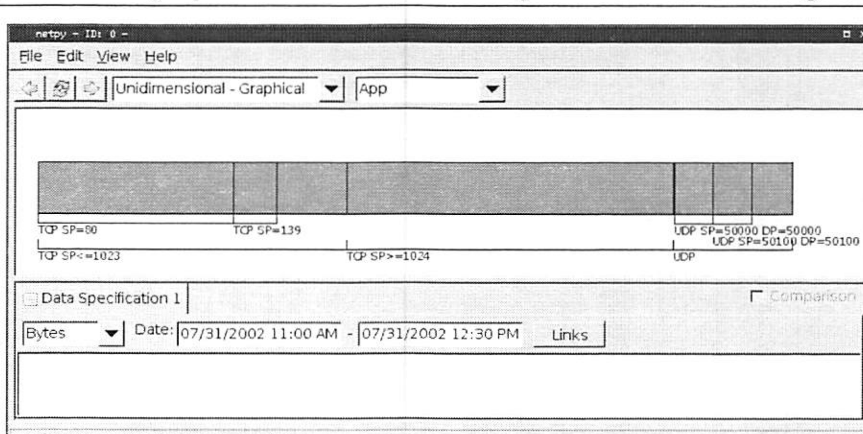


Figure 2: Hierarchical heavy hitter analysis on the application hierarchy. Based on the traffic the analysis picked to report the traffic for the entire UDP protocol, for high and low TCP source ports, for individual TCP source ports 80 (web) and 139 (netbios) and for two UDP source and destination port pairs (used by a database application).

but you won't be able to tell which one is a web server and which one is a DNS server. With Netpy's "unidimensional" reports the user can look at these hierarchies in isolation. Netpy also has "bidimensional" reports that look at two hierarchies at once: Netpy computes the relevant categories for both dimension and reports a crossproduct of the results – for every pair of categories from the opposite hierarchies, the traffic matching both categories is reported. For the example above, if we run a bidimensional analysis on the application and source address dimensions, the application dimension will pick (protocol=TCP,source port=80) and (protocol=UDP,source port=53) as relevant categories and the source address will pick (source address=A) and (source address=B). The bidimensional report will have the traffic of the following four combined categories of traffic: (protocol=TCP,source port=80,source address=A), (protocol=TCP,source port=80,source address=B), (protocol=UDP,source port=53,source address=A), and (protocol=UDP,source port=53,source address=B).

In the GUI, the two dimensions of a bidimensional report are the two sides of a square and the categories defined within individual dimensions are represented as small segments on the sides of the square. The rectangles within the square represent the combined categories. The darkness of the rectangles indicates the amount of traffic of the combined category with darker shades indicating more traffic. The GUI displays the actual amount of traffic in any combined category when the user moves the mouse over the corresponding rectangle. Using darkness to convey the intensity of traffic (or other data) is known as the "heatmap" representation. Figure 3 shows a bidimensional analysis with the application hierarchy as the horizontal dimension and the source IP address hierarchy as the vertical dimension.

The bidimensional reports do not capture all the information present in the (textual) multidimensional reports used by AutoFocus that consider all five packet header fields at the same time. The advantage of these bidimensional reports is that they can be computed much faster than the multidimensional reports and yet they can convey much of the information present in a multidimensional report.

Structure of Netpy

Netpy has four main parts: the database, the analysis engine, the console and the GUI. The role of the database is to store preprocessed NetFlow records and deliver the records selected for the current analysis to the engine. The analysis engine runs the hierarchical heavy hitter algorithm, and all other analysis algorithms supported by Netpy. The console is a text based interface to the analysis engine and the only interface that allows the network administrator to update the database. The GUI is an interface that visualizes the traffic analysis results and helps the user navigate the traffic data.

The Database

The Netpy database is an intermediary representation of the NetFlow flow records. The aim of the database is to preprocess the flow records in a way that ensures that when the user asks for an analysis, one has to read from disk the minimum amount of data needed to compute the result. Quick analyses are important for interactive exploration of the traffic mix. The entire database manipulation code is written in C and it links against the flow-tools library.

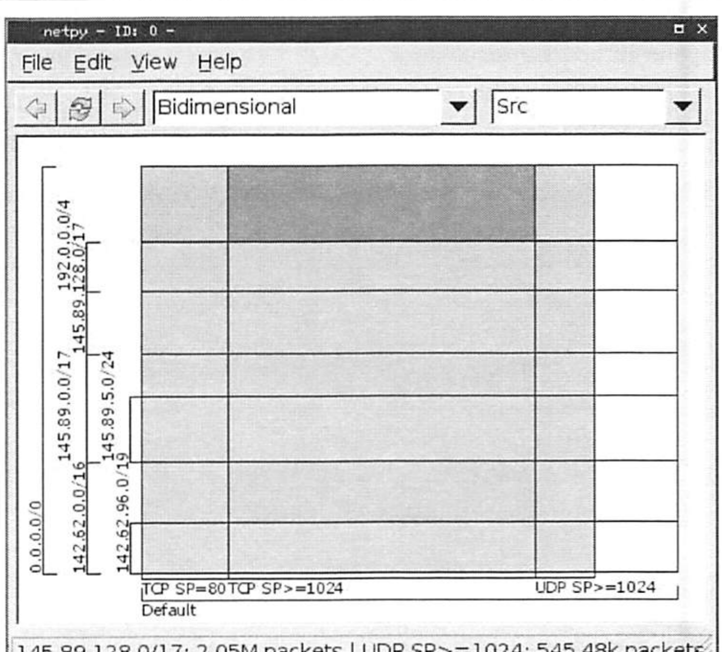


Figure 3: Bidimensional Hierarchical heavy hitter analyses on the application hierarchy and source IP address hierarchy.

Database Structure

The Netpy database is actually a hierarchy of files with simplified flow records. This format has the following four main advantages over just storing NetFlow records directly: data reduction and better control over disk usage through adaptive sampling when there are too many flow records; storing flow records for different links in different files; using separate files for time bins that make it easier to only read in the records of flows active during the selected time interval; more compact flow records with fewer fields.

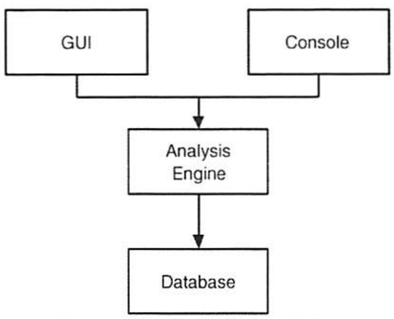


Figure 4: Netpy's modules.

Field name	Match
Exporter address	prefix match
Engine type	exact match
Engine ID	exact match
Source address	prefix match
Destination address	prefix match
Next hop addr.	prefix match
Input interface	exact match
Output interface	exact match

In the links.conf configuration file the user can define any number of links. The file has a list of rules with the NetFlow fields in this table. Each flow record is mapped to the link associated with the first rule it matches. For all fields, a '*' in the rule matches all possible values.

Netpy groups flow records into "links" based on the links.conf configuration file that uses the fields from Table 8 to select the link a flow record belongs to. This allows the network administrator to separate traffic carried on various links of a router that is nevertheless reported together. The flow records corresponding to different links are then stored in separate directories. Thus when the user runs an analysis on only one of the links, we don't have to go through all flow records, but only read those mapping to that link.

The links.conf file also specifies a cap for each link on how much hard disk space an hour's worth of traffic can take. When the number of flow records exceeds the allotted space, Netpy applies sampling using a rate that ensures that the disk usage stays within budget. The smaller the disk usage allowed, the more aggressive the sampling has to be and the less accurate the results of the analyses will be. See the section on sampling algorithms for results on the amount of error introduced by sampling.

Each directory corresponding to a link contains individual files with flow records, each representing a five minute time bin. For the current version of Netpy, this does limit the time intervals. The user can only request analyses on multiples of five minutes, but once the user specifies the interval, we can read in the right flow records by just reading the files representing the five minute bins included in the interval. Some of the original NetFlow flow records can span two or more bins. We handle these by splitting them and storing the resulting records in their respective bins. This splitting of records results in an increase of only 2% in the number of records stored in the database which is a price worth paying.

The flow records in the database contain only the fields used in the analyses: source and destination IP address, protocol, source and destination port and byte and packet count. We need not store timestamps because the file a flow record is in identifies which five minute time bin it belongs to. We also discard most of he fields from Table 8 because the useful information they hold has already been incorporated in the choice of the link the flow record is mapped to. This way the size of a flow record is reduced from 60 bytes to 21.

Reading from the Database

The analysis engine specifies what data to select for the analysis. This specification has four parts: the list of links to include, the time interval, a filter, and whether the analysis counts bytes, or packets. The first two parts of the specification determine which database files are read. As the files are read in, all records are compared against the filter and the ones not matching any rule are ignored. The records at this point represent the traffic the analysis will be run on, but the database performs two more operations to help the analysis engine: it samples and sorts the data.

Analyses that cover a large time interval and don't specify very selective filters can read millions of flow records from the database. Given that the analysis engine implements complex algorithms in python, it runs slow on this many records. Before passing the results to the analysis engine, we apply the same adaptive sampling algorithms used when writing the database to ensure that the number of flow records passed to the analysis engine is not very large (no more than 100,000 in the current version). The database also sorts the records by the field used in the analysis.

The Analysis Engine

The current version of Netpy has an analysis engine implemented entirely in python.[1] It runs the

[1]Netpy's name comes from the fact that it does network traffic analysis in python.

hierarchical heavy hitter algorithms and all other algorithms doing the analysis of the traffic. Analyses can complete in under five seconds or take as long as a minute. Running a destination address hierarchy analysis for an hour's worth of traffic takes 5.3 sec to complete. Running a application hierarchy analysis on the same time interval takes 2.7 sec. On a 24 hour time period the analysis takes 34.7 sec and 27.0 sec, for address and application analysis, respectively. We plan to reimplement most analysis algorithms in C and we expect a speedup by at least a factor of 100. It happens quite often that the user asks for the same analysis again, for example by pushing the "back" button in the interface. To avoid accessing the database and doing the computations again, the analysis engine keeps a cache of analysis results and it reads the results of old analyses out of this cache. The size of the cache is modest because the results of the analyses are typically quite small. The entries in the cache are gzipped binary dumps of the analysis data structures, the average file size is 1 KB.

The database and the analysis engine are normally part of the same process as the GUI or the console. It is also possible to run the GUI remotely and in this case the analysis engine runs as a daemon on the machine with the database.

The Console and The Graphical User Interface

The console and the graphical user interface, both implemented in python, are the two interfaces to Netpy. The console supports a simple language of commands for updating the database and performing analyses. It is the only interface that allows the user to add new NetFlow data to the database and delete old flow records. The console can accept commands interactively or as a script. The GUI, built using the wxPython user interface toolkit, visualizes the traffic analysis results and helps the user navigate the traffic data. Drill-down through clicks on graphical elements representing IP address prefixes or port ranges is integrated with filters. The "back" and "forward" buttons further facilitate the exploration of the traffic mix.

Sampling Algorithms Used By Netpy

There are two measures of traffic that Netpy analyses can choose to compute: the number of bytes, and the number of packets within various categories of traffic that make up the traffic mix. The aim of the sampling algorithms is to take a large number of flow records and reduce it to a smaller sample that is an unbiased, low error representation of the original traffic. By sampling we fundamentally loose information, so it is unavoidable that there will be errors when we estimate the traffic of some categories of traffic, and categories with little traffic are especially vulnerable.

Our aim is to pick a sampling function that ensures that the sampling error is small for the large categories of traffic. Let's focus on byte counts first. A simple solution is to sample each flow record with probability p, and for all sampled flow records to multiply their byte counts by $1/p$ to compensate for the flow records that were not selected in the sample. For example if $p = 1/5$ we would multiply by 5 the byte counts of all the sampled records. This method ensures that the number of flow records is reduced by approximately a factor of p. The errors introduced by this method are not very high if the sizes of the flows are close, but if there are a few very large flows the errors can be significant. Say the traffic mix consists of 1,000 flows of 10 KB each and one flow of 10 MB, and thus the actual total traffic is 20 MB. The sample will contain around 200 of the small flows, each counted with 50 KB of traffic so their contribution will be estimated correctly at around 10 MB, but the situation is different with the large flow: if it doesn't get sampled (and this has a probability of 80%), we don't count it at all, if it gets sampled, we count it as 50 MB. Thus we either underestimate the total traffic by a factor of $(10 + 0)/20 = 0.5$, or we overestimate it by a factor of $(10 + 50)/20 = 3$.

The solution to this problem is to use size dependent sampling, also known as smart sampling which was proposed by Duffield, et al. [4]. This methods picks the sampling probability in a way that favors the large flows. More exactly the algorithm picks a threshold z, and the flows with size $s \geq z$ are kept in the sample while the ones with $s < z$ are kept with probability $p_s = s/z$. If one of these small flows is sampled, its byte count is multiplied by $1/p_s = z/s$ which gives us a byte count of $s \cdot z/s = z$.

Smart sampling has the property that if the original set of flow records has a total traffic of T, the expected number of flow records after sampling is at most T/z, irrespective of how many flow records the original set has, and how their sizes are distributed. It also has the property that if the total traffic of a category is C the standard deviation (average error) of the estimate of the traffic of the category after sampling is at most $\sqrt{Cz}$, but it can be smaller depending on the distribution of the sizes of the flows that are part of the category. For example if the total traffic is $T = 100{,}000$ MB, and we use a threshold of $z = 1$ MB, the number of flow records in the sample is expected to be at most 100,000 (if the original traffic mix has many flows significantly larger than 1 MB, the number of flow records in the sample will be significantly below 100,000). If we want to estimate the total traffic T based on the sample, the standard deviation of the result will be at most $\sqrt{Tz} = 316$ MB and the the probability that we overestimate or underestimate the total traffic of 100,000 MB by more than $3\sqrt{Tz} = 948$ MB (an error of less than 1%) is below 0.3%. If we look at a smaller category with a traffic of $C = 10{,}000$ MB, the probability that we underestimate or overestimate its traffic by more than $3\sqrt{Cz} = 300$ MB (an error of

3%) is below 0.3%. The larger z, the smaller the sample size, the larger the errors. If we increase z by a factor of 100, we reduce the sample size by a factor of 100, but we increase the errors by a factor of 10.

When adding data to the database Netpy uses the limit imposed on the disk usage of the database to indirectly determine the threshold z. For example if the limit for one hour's data is set to 10 MB in links.conf, this translates to 853 KB for each of the 12 files representing a five minute bin, and since the size of a flow record is 21 bytes this translates to 41,600 records. Thus the value of z will be at most one 41,600th of the total traffic for the five minutes. This translates to a standard error for the estimate of the total traffic during those five minutes of at most $\sqrt{1/41,600} = 0.49\%$. If we look at a smaller category of say 1% of the traffic during those five minutes the standard error of the estimate of the components traffic is at most 4.9%. Of course, if one looks at longer time periods (an hour, or a day) since there will be more flow records, the relative errors in the estimates will go down.

The sampled flow records used for estimating byte counts can also be used for estimating packet counts. Since the sampling is biased towards flows with many bytes it will also catch flows with many packets. Packet sizes vary between 40 bytes and 1500 bytes so it will happen that a flow with fewer larger packets will be preferred over a flow with more but smaller packets, but since the ratio between the largest and the smallest packet is only 37.5 the types of pathological errors that are possible with uniform record sampling are not possible. Let s' be the number of packets in a flow record. If $s \geq z$, the flow record will be sampled and s and s' will remain unchanged. If $s < z$ and the flow record is sampled we multiply the packet count by $1/p_s$ and thus have a packet count of $s' \cdot z/s$ in the flow record we keep in the sample. The errors in the estimates for the number of packets in various categories of traffic are similar to the errors in the byte counts.

Future Work

While Netpy is the result of a lengthy design and development process, we plan to improve it further. A first thing to do is to add flow counting functionality (partially implemented) because flow counts are better at revealing many types of traffic a security-conscious network administrator might want to know about such as scans and floods with source addresses spoofed at random. We can group the improvements we plan into improvements that will increase the speed of the analyses, and improvements that will increase their power.

The current performance bottleneck is the hierarchical heavy hitter algorithms in the analysis engine. We plan to reimplement all hierarchical heavy hitter algorithms in C and based on measurements of AutoFocus' C backend that runs similar algorithms we expect a speedup by at least a factor of 100. The database read can also become a bottleneck when reading large amounts of data (e.g., running an analysis on an entire month). We plan to address this performance bottleneck by conceptually keeping multiple versions of the database a very small one with very aggressive sampling, a medium one and a large one with mild sampling. An analysis on long time intervals would use the coarsest database to reduce the amount of disk reads, while one on short time scales would use the most detailed one to get accurate results. In practice we can integrate these multiple conceptual databases into a single file hierarchy. More compact encodings of flow records or the use of gzip to compress the flow record files will reduce disk usage and increase performance since less data will have to be read.

There are a few directions in which we plan to increase the power of Netpy's analyses. Due to the five minutes bins used currently the analysis cannot look at a granularity finer than five minutes, even though the original NetFlow data would have allowed it. By adding small timestamps to the flow records we hope to be able to support analysis at the granularity of seconds. Another direction of improvement relies on the observation that while currently all analyses work on a portion of the traffic mix, it often makes sense to compare the current traffic against historical traffic to find the things that have changed. We plan to extend Netpy with "comparison reports" working on two data sets at a time. A third direction plans to address limitations due to the fact that the analysis of IP addresses relies on the implicit hierarchy in the IP address space. The problem with this approach is that because of how the IP address space is allocated, prefixes are not always meaningful, they can include portions of unrelated organizations. We plan to extend Netpy with three more hierarchies for IP addresses: one based on DNS reverse mappings of IP addresses, one based on whois data, and one based on BGP routing table information. Hierarchical heavy hitter algorithms can be easily adapted to all of these hierarchies and they can provide valuable new insights into the traffic mixes on our networks.

Our hope is that many of these improvements will be implemented and mature enough for widespread use by the end of the 2005.

Conclusions

IP traffic can be unpredictable and traffic analysis can help with incident response as well as with long term planning of network growth. This paper presents Wisconsin Netpy, an application for analyzing and visualizing NetFlow traffic data. While Netpy is certainly not the first application in this space, we believe that it incorporates important new ideas that enable powerful exploratory analyses of the traffic mix not supported by other tools currently used by network administrators. The use of a small database of sampled

traffic enables prompt analyses that allow the network administrators to refine their queries iteratively. The unidimensional and the novel bidimensional hierarchical heavy hitter analyses can provide a detailed view of the traffic. The visualization of analysis results in the form of time series plots and heatmaps makes it easier for a human observer to absorb information about the composition of the traffic mix. We hope that Netpy will contribute to a better understanding of how networks are used and through this understanding to better managed networks that offer more reliable service to millions of Internet users worldwide.

Acknowledgments

We thank John Henry, Fred Moore, Jaeyoung Yoon, Brian Hackbarth, Ryan Horrisberger, Pratap Ramamurthy, Dan Wendorf, Steve Myers, and Dhruv Bhoot who were part of the teams that worked on Netpy as a class project in Fall 2004 and 2005. Early conversations with Glenn Fink and Chris North at Virginia Tech lead to the use of heatmaps as visualization metaphor. We thank Mike Hunter for suggestions for features in Netpy and Dave Plonka for providing us with generous amounts of NetFlow data to test on and valuable feedback.

Author Information

Cristian Estan graduated from the Technical University of Cluj-Napoca, Romania in 1995 with a degree in Computer Science. His first real job, was as network/system administrator, and it taught him that configuring software or networking gear always takes longer than expected. After moving to the U. S. in 1998 he worked at two startups and eventually managed to get a Ph.D. at U. C. San Diego. Currently he is an assistant professor at U. W.-Madison and can be reached at estan@cs.wisc.edu .

Garret Magin graduated from the University of Wisconsin-Madison in May of 2005 with degrees in computer science, computer engineering, and math. He recently started working in the embedded networking space on the Windows CE core networking team. When he is not at work and its not raining he is off riding his Honda Superhawk. He can be reached at garret.magin@microsoft.com.

Bibliography

[1] Allen, Jeff R., "Driving by the rear-view mirror: Managing a network with cricket," *USENIX 1st Conference on Network Administration*, April, 1999.

[2] *cflowd: Traffic flow analysis tool*, http://www.caida.org/tools/measurement/cflowd/ .

[3] Cormode, Graham, Flip Korn, S. Muthukrishnan, and Divesh Srivastava, "Finding hierarchical heavy hitters in data streams," *VLDB*, December, 2003.

[4] Duffield, Nick, Carsten Lund, and Mikkel Thorup, "Charging from sampled network usage," *SIGCOMM Internet Measurement Workshop*, November, 2001.

[5] Estan, Cristian, "Autofocus: A tool for automatic traffic analysis," *29th meeting of NANOG*, October 2003.

[6] Estan, Cristian, Stefan Savage, and George Varghese, "Automatically inferring patterns of resource consumption in network traffic," *Proceedings of the ACM SIGCOMM*, August, 2003.

[7] Fullmer, Mark, and Steve Roming, "The OSU flow-tools package and cisco netflow logs," *USENIX LISA*, December, 2000.

[8] Gates, Carrie, Michael Collins, Michael Duggan, Andrew Kompanek, and Mark Thomas, "More netflow tools for performance and security," *USENIX LISA*, November, 2004.

[9] Kaizaki, Ryo, *Aguri: An aggregation-based traffic profiler*, http://www.csl.sony.co.jp/person/kjc/kjc/software.html#aguri .

[10] Lakkaraju, Kiran, William Yurcik, and Adam J. Lee, "Nvisionip: Netflow visualizations of system state for security situational awareness," *ACM VizSEC/DMSEC04*, October, 2004.

[11] Lau, Stephen, "The spinning cube of potential doom," *Communications of the ACM*, Vol. 47, June, 2004.

[12] Oetiker, Tobias, "Mrtg – the multi router traffic grapher," *USENIX LISA*, December, 1998.

[13] Plonka, David, "Flowscan: A network traffic flow reporting and visualization tool," *USENIX LISA*, pages 305-317, December, 2000.

NetViewer: A Network Traffic Visualization and Analysis Tool

Seong Soo Kim and A. L. Narasimha Reddy – Texas A&M University

ABSTRACT

The frequent and large-scale network attacks have led to an increased need for developing techniques for analyzing network traffic. If efficient analysis tools were available, it could become possible to detect the attacks, anomalies and to appropriately take action to contain the attacks before they have had time to propagate across the network. This paper describes NetViewer, a network monitoring tool that can simultaneously detect, identify and visualize attacks and anomalous traffic in real-time by passively monitoring packet headers. NetViewer represents the traffic data as images, enabling the application of image/video processing techniques for the analysis of network traffic.

NetViewer is released free to the general public. By employing a freely available visualization tool, the users of NetViewer can comprehend the characteristics of the network traffic observed in the aggregate. NetViewer can be employed to detect and identify network anomalies such as DoS/DDoS attacks, worms and flash crowds. NetViewer can also provide information on traffic distributions over IP address/port number domains, utilization of link capacity and effectiveness of Quality of Service policies.

Introduction[1]

The frequent and increasing malicious attacks on network infrastructure, using various forms of denial of service attacks, have led to an increased need for developing techniques for analyzing network traffic. If efficient analysis tools were available, it could become possible (i) to detect the attacks, anomalies and (ii) to appropriately take action to mitigate the attacks before they have had time to propagate across the network or to cripple the infrastructure. These tools may be in turn useful for traffic engineering purpose since the network traffic analysis provided by these tools could lead to the identification of resource bottlenecks and peak usage.

A variety of tools for flow-based measurement have arisen from both the commercial and free software communities. To study and classify traffic on the network based on usage and protocols, a number of tools such as FlowScan [PLO00], Cisco's FlowAnalyzer, and AutoFocus [ESTsv03], are used as traffic analyzers. While flow-based features within the network infrastructures are convenient, such approaches may not be sufficient for reliable and fast application. Some of these tools provide real-time reporting capability, but much of the analysis is done off-line. These tools have been effectively utilized for traffic engineering and postmortem anomaly detection.

However, rigorous real-time analysis is needed for detecting and identifying the anomalies so that mitigation action can be taken as promptly as possible. Some of these tools are based on the volume of traffic such as byte counts and packet counts. When links are not sufficiently provisioned, normal traffic volumes may reach the capacity of the links most of the time. In such cases, attack traffic may not induce significant overshoot in traffic volume (merely replacing existing normal traffic) and hence may make traffic volume signal ineffective in detecting attacks.

Sophisticated low-rate attacks [EUZk03] and replacement attacks, which don't give rise to noticeable variance in traffic volume, could go undetected when only traffic volume is considered. Furthermore, the tools which collect and process flow data may not scale to high-speed links as they focus on individual flow behavior. Our tool tries to look at aggregate packet header data in order to improve scalability.

Intrusion detection systems (IDS) such as Snort and Bro are an important part of network security architecture and signature database-based monitoring of network traffic for predefined suspicious activity or patterns. These tools are widely deployed by network administrators. This detection principle relies on the availability of established rules of the anomalous or suspicious network traffic activity. While the identification mechanism of the IDS tools provides fine-grain control of network flows, they however need to be updated continuously with the latest rules for coping with novel attacks. Our approach tries to develop a generic mechanism independent of specific anomalies for improving adaptability.

In this paper, we describe a tool named NetViewer for traffic anomaly detection based on analyzing the distribution of traffic header data, in

[1]This work is supported by NSF grants ANI-0087372, 0223785, Texas Higher Education Board, Texas Information Technology and Telecommunications Taskforce and Intel Corp.

postmortem and in real-time. We adopt a network measurement-based approach that can simultaneously detect, identify and visualize attacks and anomalous traffic in real-time. We propose to represent samples of network packet header data as frames or images.

With such a formulation, a series of samples can be seen as a sequence of frames or video. This enables techniques from image processing and video compression such as scene change analysis and motion prediction to be applied to the packet header data to reveal interesting properties of traffic. Our work here brings techniques from image processing and video analysis to visualization and real-time analysis of traffic patterns.

We show that "scene change analysis" can reveal sudden changes in traffic behavior or anomalies [LELs03, ZHAks93, LIEke97, SHEd95, GYAkc03]. We show that "motion prediction" techniques can be employed to understand the future patterns of some of the attacks. We show that it may be feasible to represent multiple pieces of data as different colors of an image enabling a uniform treatment of multidimensional packet header data. NetViewer can give an intuitive and descriptive illustration of network traffic, with visible features, to network operators.

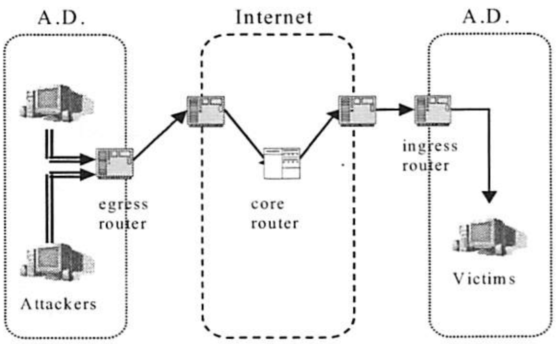


Figure 1: The installation locations of NetViewer. For ingress/egress filtering, it can be employed in a router or at the boundary of an administrative domain.

Environment

The visibility of fields in the network packets may be impacted by the location where traffic is observed. NetViewer can be applied for analyzing inbound/outbound traffic at administrative domain (AD) edge router as shown in Figure 1. The AD represents the access link, ISP (Internet Service Provider) and intra-domain IP-based networks such as enterprise networks and campus networks.

Employing ingress filtering with NetViewer monitors the flow of traffic as it enters a network under administrative control. NetViewer can be combined with a Quality of Service (QoS) policy framework to rate limit high-bandwidth flows identified by NetViewer.

Egress filtering with NetViewer inspects the flow of traffic as it leaves a network under administrative control. There are typically policy regulations for inner machines initiating outgoing connections to the Internet. Outbound filtering has been advocated for limiting the possibility of address spoofing, i.e., to make sure that source addresses correspond to the designated addresses for the AD. Traffic monitoring at a source network enables a detector to detect attacks early, to control hijacking of AD machines, and to limit the liability from such attacks and the squandering of resources. NetViewer can help prevent compromised systems on a network from attacking systems elsewhere through egress filtering.

Our traffic attack/anomaly filtering efforts have been targeted at the edge environment due to resource utilization issues and source address spoofing concerns.

Goals

NetViewer's goal is not to detect and eliminate the anomalies eradicatively, but to bring the anomalous traffic under control in real-time so that the mitigation mechanisms could be deployed fast to counter the threats of the anomalies. We also want NetViewer to not contain any legitimate traffic in the process of containing attack traffic. Our goal is to detect and contain 99.7% of the attack and anomalous traffic at a false alarm rate of 0.3%. Given the current and growing network link speeds, anything we implement needs to be fast and efficient enough to not excessively burden existing network infrastructure.

As an operational goal, we want that the containment is centrally processed at a router level, rather than something that the end users would have to deal with. Netviewer can work with packets on the fly at a router in real-time mode and packet traces in libpcap or NetFlow format in post-mortem mode.

Architecture

NetViewer's architecture is mainly focused on performance, simplicity, and versatility. NetViewer system's architecture consists of five major software components: the packet parser, the signal computing engine, the detection engine, the visualization engine and the alerting engine, which are programmed in ANSI C and Matlab language [MAT01].

The Packet Parser

The packet parser engine is responsible for collecting and processing raw packets and traffic data exported from routers. NetViewer can work with traffic records in postmortem or work with more aggregate data upon packet arrival in real-time. It can parse packets on the fly and parse network packet header traces with libpcap (packet capture library) [PCAP94], Cisco's NetFlow [NETF] and NLANR formats [NLA02] such as DAG, Coral, and TSH (time sequenced headers).

Traffic volume, such as packet counts, byte counts and the number of flows, can be used as a

signal, and fields in the packet header, such as addresses, port numbers and protocols, can be employed as an observed domain. According to operator's concern, NetViewer then generates images of the distribution of traffic intensity in the chosen domain. Based on the kinds of traffic data and the header domain, we categorize the image-based signals into address-based, flow-based and port-based signals. Address-based signal employs packet count distribution over address domain (either source address alone, or destination address alone, or a 2-dimensional source and destination address domain). Flow-based signal employs the flow number distribution over address domain(s). Port-based signal employs packet count distribution over port number domain.

Traffic headers such as addresses and port numbers have larger spaces over which data is distributed, that is, 2^{32} IPv4 addresses and 2^{16} port numbers. Any developed technique should be simple enough to be deployable, i.e., should not be expensive in terms of memory and processing resources. In order to address the problem of large domain spaces, we have employed a concise data structure for reducing the domain space [KIMrv04]. We explain this data structure using address-based signal as an example. This data structure *count*[*i*][*j*][*t*] represents the data sample at time *t*.

The data structure consists of four arrays *count*[4] for the 4 bytes of the IP address. Within each array, we have 256 locations, for a total of 4*256 locations = 1024 locations. By default, the size of one location is set to 16 bits. A location *count*[*i*][*j*][*t*] is used to record the packet count for the address *j* in *ith* field of the IP address in time interval *t*. This provides a concise description of the address instead of 2^{32} locations that would be required to store the address occurrence uniquely. Upon packet arrival, the corresponding four positions of the data structure are updated through scaling.

The Signal Computing Engine

Each sampling period, the packet counts of the entire traffic are recoded to the corresponding positions of each IP address byte-segment, and the normalized packet count is quantized and represented using Equation (1). Each resultant normalized packet count then represents the intensity of the corresponding pixel in the image representation of the traffic as shown in Figure 2.

$$p_{ijt} = \frac{count[i][j][t]}{\sum_{j=0}^{255} count[i][j][t]}, \quad \begin{matrix} i = 0, 1, 2, 3 \\ j = 0, \ldots, 255 \end{matrix} \qquad (1)$$

In order to quantitatively analyze the network traffic anomalies, we compute correlation and deltas based on normalized packet counts. Consider two adjacent sampling instants. We can define correlation signal at sampling point *t* by (2-1), which measures correlation of traffic intensity at a particular address. Delta is defined as the difference of normalized packet counts by (2-2), which is a useful signal at the beginning and ending of attacks.

$$C_{ijt} = p_{ijt} \times p_{ijt-1} \qquad (2\text{-}1)$$
$$\Delta p_{ijt} = p_{ijt} - p_{ijt-1} \qquad (2\text{-}2)$$

We employ the variance of pixel intensities in the image as traffic signal for scene change analysis, which is denoted by S_σ. Using the variance of these image signals for deriving thresholds, we can obtain an approximation of the energy distribution of the normalized packet counts within the observation domain as follows:

$$S_\sigma = \sqrt{\frac{1}{1024}\sum_{i=0}^{3}\sum_{j=0}^{255}(p_{ijt} - \bar{p}_{ijt})^2} \qquad (3)$$

where p_{ijt} are pixel intensities and

$$\bar{p}_{ijt} = \frac{1}{1024}\sum_{i=0}^{3}\sum_{j=0}^{255} p_{ijt}.$$

Upon each sampling instant, the aggregate traffic signal is instantaneously calculated based on accumulated data structure for real-time analysis.

The Detection Engine

The detection engine employs a theoretical basis for deriving thresholds for analyzing traffic signals and anomaly detection. For 3σ-based statistical analysis, we set two kinds of thresholds, a high threshold T_H and a low threshold T_L. When we respectively set the T_H and T_L thresholds to $\pm 3\sigma$ of aforementioned traffic signal distributions in ambient traffic, attacks can be detected with an error rate of 0.3% (if the signal is normally distributed) which can be expected as target false alarm rate as (4-1) [NIS05]. For deriving initial thresholds from background traffic, a tune-up procedure is necessary just after powering up. By default, it is set to 120 samples; that is 2 hours in case of 1 minute samples. To analyze the statistical properties of normal traffic dynamically, we employed an exponential weighted moving average (EWMA) of normal traffic free of attacks. The dynamic average of the traffic is updated at every sampling point excluding attack periods.

The detection engine can judge the current traffic status by calculating the standard intensity deviation of signals in each sampling instant by (4-2). The analyzed information will be compared with historical thresholds of traffic to see whether the traffic's characteristics are out of regular norms. Sudden changes over 3σ in the analyzed signal are expected to indicate anomalies.

$$X \sim N(\mu, \sigma^2) \rightarrow \Pr(\mu - 3\sigma < X \le \mu + 3\sigma) \approx 99.7\% \qquad (4\text{-}1)$$

$$traffic\ status \begin{cases} normal, & \text{if } T_L < S_\sigma < T_H \\ attack, & \text{if } S_\sigma \le T_L \ or \ T_H \le S_\sigma \end{cases} \qquad (4\text{-}2)$$

Also, with mean, standard deviation and signal value at every sampling point, the detection engine would compute the probability of anomaly assuming the normal distribution [KILn02, NIS05].

The Visualization Engine

The visualization engine employs graphic library of Matlab (in this implementation) for displaying

traffic signals and images. The visualization engine produces user-friendly, images of network traffic. As shown in Figure 2, the visual parts of NetViewer's main screen make up four primary components: the general traffic profile, traffic distribution signals, traffic images and anomaly reports. While the profiling and reporting components are expressed in text, the traffic signals and images are visualized in graphs. The visualization engine enables NetViewer to offer these visual measurements as a real-time motion picture. It could help the network operators recognize the trends and transitions in network traffic.

The visualization engine plots the standard deviation of traffic intensity computed by (3) versus the sampling instant in source and destination domains respectively as network traffic signals. If the anomaly is detected, a red dot is marked on the bottom and calls the network operator's attention. NetViewer's viewing window is controllable for short-term and long-term analysis purposes. By default, the window is 60 sampling points, which is the latest 1 hour with 1 minute samples.

Each element of the data structure computed by (1) corresponds to a rectangular area (or pixel) in the traffic image. The values of the elements of the data structure are indexed into the current color-table that determines the color of each pixel. The color of each pixel shows the intensity of traffic at the source or destination or (source, destination) pair in 2-dimensions.

The descending order of intensity is black, red, orange, yellow and white. Each quadrant corresponds to each byte in IP address structure. In 1-dimensional source or destination domain, each quadrant consists of 16 by 16 pixels for mapping 256 elements of one byte of IP address as shown in Figure 3(a). Each quadrant maps the 0 to 255 values of one byte of IP address in a row-major order. Thus four quadrants consist of the entire 4-byte IP address. The four bytes 0 to 3 of IP address are also organized as quadrants in a row-major order. In 2-dimensional image, the x-axis in each quadrant corresponds to the distribution of the destination IP addresses, and y-axis that of the source addresses as shown in Figure 3(b). In each quadrant, source and destination addresses consist of 256 by 256 pixels.

The Alerting Engine

Once anomalies are detected through scene change analysis, the alerting engine scrutinizes the

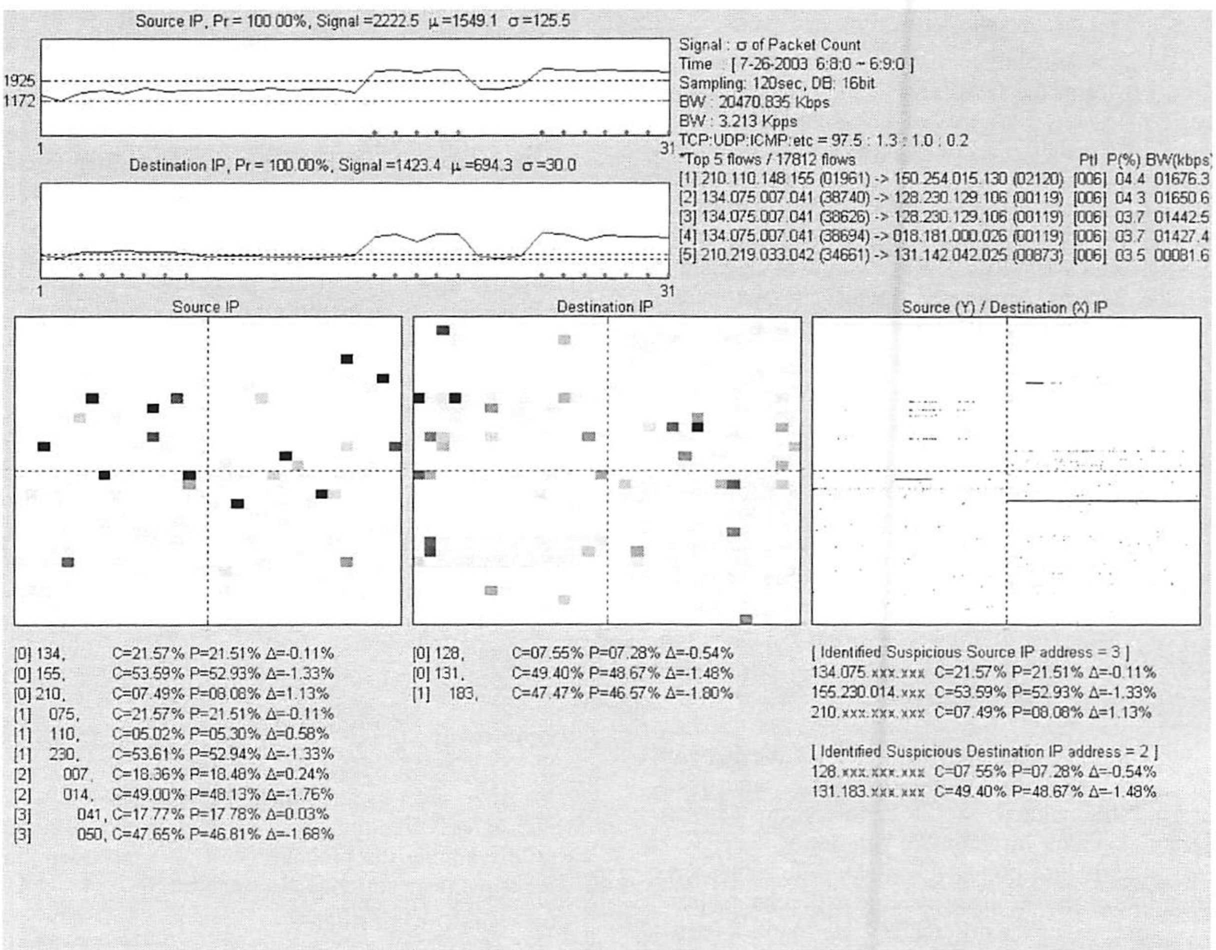


Figure 2: A running example of NetViewer.

above quantities by (1) and (2) for identification purposes. From the predefined correlation and delta thresholds, the alerting engine can identify the IP addresses of suspicious attackers and victims. Based on the revealed IP addresses, we closely investigate each address on the basis of statistical measurements. This inspection will lead to some form of a detection signal that could be used to alert the network administrator of the potential anomalies in the network traffic. The alerting engine generates the detection reports in online image as well as in an offline file.

NetViewer's Functionality

NetViewer and its component engines are responsible for profiling and monitoring raw packets or trace data exported from routers. NetViewer monitors each packet and maintains data structures based on the observed domain (address, port numbers etc.). When anomalies are detected, NetViewer reports its detection results and may optionally take containment actions. It may be configured to either archive for postmortem analysis or discard the counter contents after processing.

Controllable parameters, such as window size for determining the amount of retained data and thresholds, can be configured before or after the packet parser engine takes an action. By default, a sampling interval is set to 60 seconds for deriving most stable traffic images, and a sampling duration ratio is 1:1. That is to say, we sampled for 30 seconds and paused for 30 seconds for reducing the processing requirements. Our techniques are light-weight enough for traffic to be continuously sampled without any pause periods.

Figure 2 is a sample NetViewer graph of real traffic with attack at an access link.

Traffic Profiling Function

The upper right text in Figure 2 shows the general information of current network traffic. The profile includes the selected signal type, the local time, the selected sampling period, number of bits used to represent traffic intensity of individual pixels in the data structure, and the bandwidth in Kbps (bits per second) and Kpps (packets per second).

The next line illustrates the proportion occupied by each traffic protocol. It is possible to determine whether the current traffic is behaving normally through correlating it to that of previous states of traffic from a protocol viewpoint. This is based on the observation that during the attacks, the protocol employed by the attack traffic should see considerably more traffic than during normal traffic.

The Top 5 flows term shows the topmost five flows out of the total flows in the selected traffic volume, which can be packet count, byte number or the number of flows. It is expressed as a quintuple of source IP address/port number and destination IP address/port number, exploited protocol, occupied proportion in percentage, and bandwidth in Kbps. It is implemented using LRU (Least Recently Used) policy with partial state for identifying long-term high-rate flows [SMIkr01].

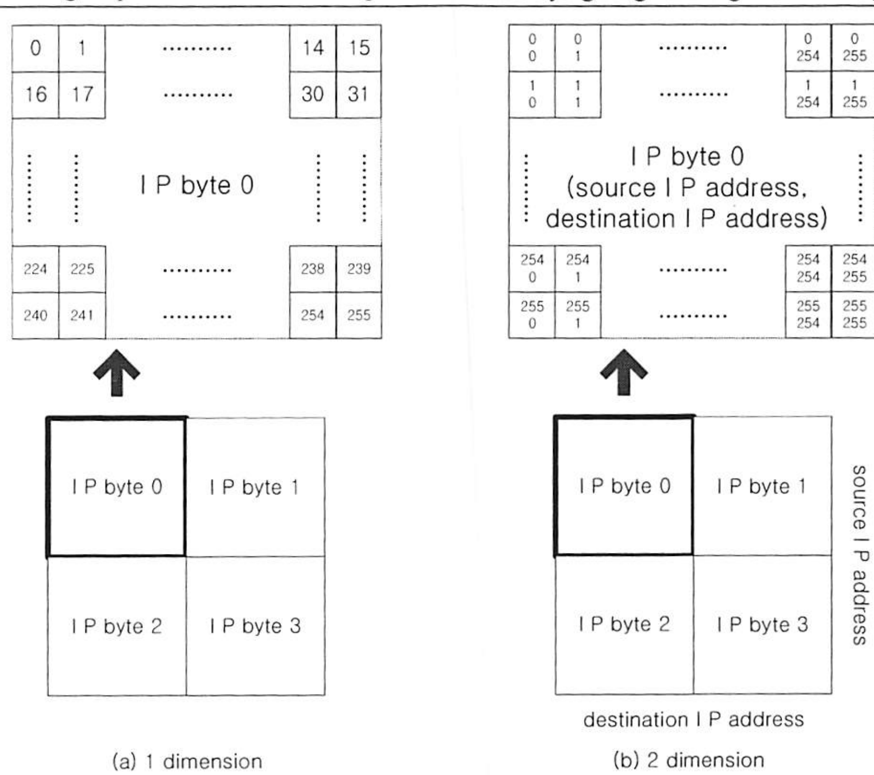


Figure 3: The visualization of the network traffic signal in the IP address domain.

The profiling information presented by NetViewer assists in understanding the general nature of the traffic at the monitoring point.

Monitoring Function

The two upper left sub-pictures in Figure 2 illustrate traffic distribution signal over the latest predefined (and adjustable) time-window. These two pictures map to source domain and destination domain respectively. The captions above each picture explain the current traffic distribution. The Source IP term means that this signal is originated from packet counts in the source IP address domain. Based on the operator's selection, this term can be changed to Source FLOW which analyzes the number of flows over the source IP address domain, or to Source PORT which analyzes packet counts in the source port domain, or to Source MULTIDIMENSIONAL which analyzes multiple components of different distributions in the source IP address domain.

The Pr term estimates the anomalous probability of current traffic distribution assuming Gaussian distribution. The probability is computed from the normal distribution table based on the traffic signal, its μ and σ [NIS05]. We can be informed the possibility of anomalous traffic quantitatively. The Signal term is computed by (3). The "μ" and "σ" terms mean the mean value and the standard deviation of distribution signal using EWMA. The left two figures in y-axis and dotted vertical lines illustrate the $\pm 3\sigma$ levels respectively. The above statistical measurements are dynamically updated at every sampling point excluding attack periods.

The red dots located on the bottom of the each sub-picture are marked when 3σ-based statistical analysis detects anomalies. The detection signal automatically triggers to identify the IP addresses of sources and destinations and can be used to alert traffic anomalies to the network operator.

Anomaly Reporting Function

The center three sub-pictures in Figure 2 illustrate image-based traffic in the source/destination IP address domain and the 2-dimensional domain.

The color and darkness of each pixel indicate the intensity of traffic of corresponding IP address. In case of normal traffic, the aggregate traffic does not form any regular shape due to dispersibility of traffic of various and numerous flows in time and space. In the case of abnormal traffic, however, the traffic pattern of network may change and these changes could be exhibited in the visual images. From destination IP address in Figure 2, a specific area of IP byte2 is shown in a darker yellow shade. It illustrates that the current traffic is concentrated on a (aggregated) single destination or a

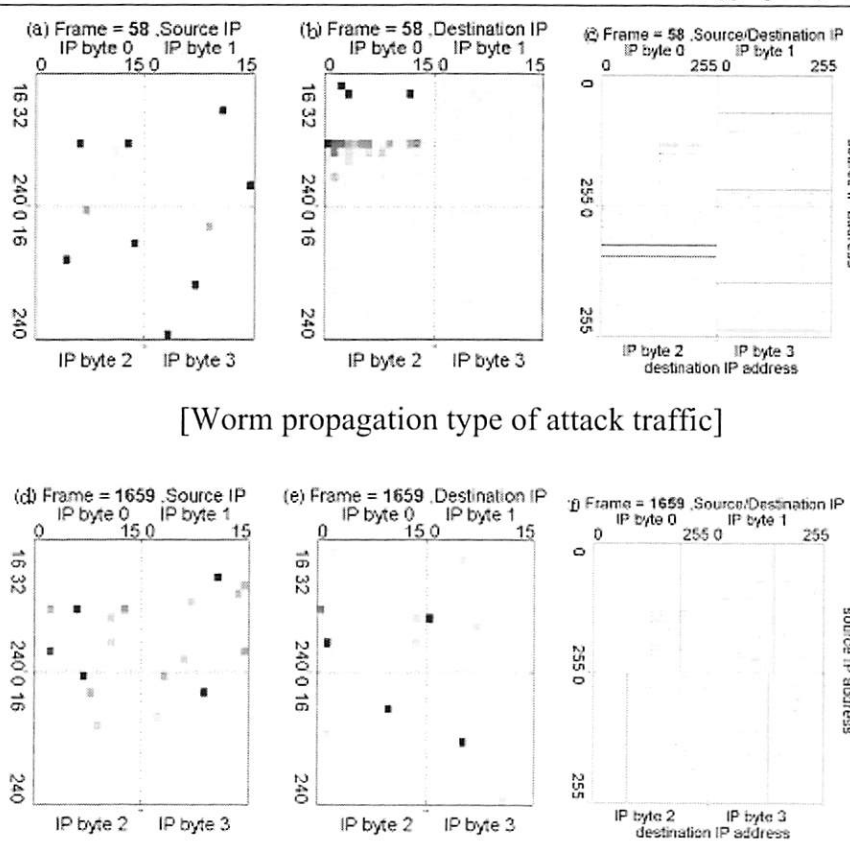


Figure 4: Various attack images reported by NetViewer.

subnet. NetViewer also shows that a specific source, i.e., an attacker, monopolizes network traffic, shown in the form of a line in the 2-dimensional domain.

The exposed images can show various kinds of patterns according to the nature of attacks. Figure 4 shows the visual expression of a worm and a DDoS attack respectively. While the horizontal lines imply that the same source targets multiple destinations in a worm attack, the vertical lines reveal that several machines (in a subnet) are targeting a single server in a DDoS attack.

Once anomalies are detected, the identified IP addresses of sources and destinations are revealed in byte-segment level and 4-byte whole structure simultaneously as shown in the lower text of Figure 2. These identified IP addresses are quantitatively investigated on the basis of statistical measurements using the correlation (C), the possession ratio (P), the delta (Δ) between consecutive frames and the black list (S) computed by the signal computing engine. S recorded in the last column indicates black listing which is successively identified and refined over recent sampling instances. It could help network operators make a final decision.

Additionally the detection report is optionally generated in a file format. The upper part of Figure 5 shows the identified addresses responsible for the anomalies over the four byte-segment levels and the lower part of Figure 5 illustrates source and destination addresses of attacks.

Auxiliary Function

NetViewer could support various auxiliary functions. The video/frame representation of network traffic enables motion estimation based techniques for attack prediction. Unassigned address ranges can be clearly marked in the generated images to identify address spoofing. We describe some of these functions here.

Multidimensional Image

Up to now, we have focused on address-based image on which normalized packet counts are visualized in the address domain. By the operator's selection, NetViewer's main screen can be changed to flow-based image which visualizes the number of flows over the address domain, or to port-based image which displays packet counts in the port domain, or to multidimensional

```
*******************************************************************
[ Time : Sat 07-26-2003 05:12:00 ]
------------------------------------------------------------
Source IP[0]   134.       correlation = 17.48%  possession = 18.77%  delta = 2.50%  S
Source IP[0]   141.       correlation =  4.33%  possession =  3.94%  delta = 0.79%  S
Source IP[0]   155.       correlation = 58.20%  possession = 56.80%  delta = 2.84%  S
Source IP[0]   210.       correlation =  5.66%  possession =  6.51%  delta = 1.60%  S
Source IP[1]      75.     correlation = 17.47%  possession = 18.77%  delta = 2.51%  S
Source IP[1]     110.     correlation =  4.62%  possession =  5.25%  delta = 1.21%  S
Source IP[1]     223.     correlation =  4.31%  possession =  3.94%  delta = 0.78%  S
Source IP[1]     230.     correlation = 58.21%  possession = 56.84%  delta = 2.76%  S
Source IP[2]         7.   correlation = 15.59%  possession = 17.02%  delta = 2.74%  S
Source IP[2]        14.   correlation = 53.99%  possession = 52.31%  delta = 3.41%  S
Source IP[3]          41  correlation = 15.16%  possession = 16.36%  delta = 2.30%  S
Source IP[3]          50  correlation = 52.58%  possession = 50.83%  delta = 3.54%  S
------------------------------------------------------------
Identified No. 1st = 4, 2nd = 4, 3rd = 2, 4th = 2
===================================================================
Destination IP[0] 18.       correlation =  4.37%  possession =  3.88%  delta = 1.01%  S
Destination IP[0] 128.      correlation =  6.08%  possession =  7.01%  delta = 1.75%  S
Destination IP[0] 131.      correlation = 53.65%  possession = 52.33%  delta = 2.67%  S
Destination IP[1]    181.   correlation = 56.03%  possession = 54.00%  delta = 4.15%  S
Destination IP[3]        26 correlation =  3.89%  possession =  3.58%  delta = 0.65%  S
------------------------------------------------------------
Identified No. 1st = 3, 2nd = 1, 3rd = 0, 4th = 1
===================================================================
* Identified Suspicious Source IP address(es)
        134. 75. 7. 41  correlation = 17.48%  possession = 18.77%  delta = 2.50%  S
        141.223.xxx.xxx correlation =  4.33%  possession =  3.94%  delta = 0.79%  S
        155.230. 14. 50 correlation = 58.20%  possession = 56.80%  delta = 2.84%  S
        210.xxx.xxx.xxx correlation =  5.66%  possession =  6.51%  delta = 1.60%  S
------------------------
* Identified Suspicious Destination IP address(es)
        18.xxx.xxx.xxx  correlation =  4.37%  possession =  3.88%  delta = 1.01%
        128.xxx.xxx.xxx correlation =  6.08%  possession =  7.01%  delta = 1.75%  S
        131.181.xxx.xxx correlation = 53.65%  possession = 52.33%  delta = 2.67%
*******************************************************************
```

Figure 5: The detection report of anomaly identification (with anonymized IP addresses).

image which simultaneously exhibits multiple components of different images in the address domain.

In multidimensional images, individual components, such as packet counts, the number of flows and the correlation of the packet counts, can be represented as different components (for example, Y, U, V) of an image or different primary colors (R, G, B). With the three distinct traffic components, NetViewer can comprehensively analyze the traffic properties of each IP address from diverse viewpoints.

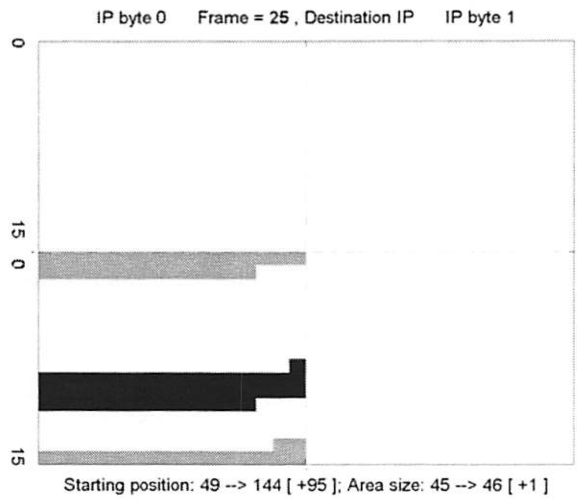


Figure 6: Potential attack estimation in red using motion prediction.

Attack Tracking

During some attacks, a concentrated attack is circulated on the address space. Using motion prediction, it is possible to expect or anticipate the next set of target addresses in such attacks. NetViewer estimates the locations of the next attack using modified motion prediction scheme. The results from such an analysis on a predictable attack are shown in Figure 6. Figure 6 shows the ongoing attacked parts in black and the result of motion prediction (in red pixels) indicating the next set of addresses that may be a target of this attack. The next potential attack ranges are estimated based on the starting positions of current attack and the motion vector length. The attack tracking would be displayed on NetViewer's subsidiary screen.

Automatic Spoofed Address Masking

In bandwidth consumption attacks such as traditional flood-based attacks, the source IP address of the packet could be usually spoofed through the abuse of raw sockets function or the like. The deliberate falsification is often forged in random order or in a dictionary order. Random destination addresses may also be employed during probing for possible infiltration. SQL Slammer was the representative random propagation worm [CER03]. And the portscan-based attacks heavily exploit the destination port numbers in random or sequential order.

Global address allocations are organized by types, which are classified according to prefixes. Currently, many portions of the IP address space are still unassigned by IANA (Internet Assigned Numbers Authority) [IPV04]. Especially, the unallocated address space can be clearly identified in the first byte image in NetViewer. Moreover, if NetViewer is located at the boundary of the AD, the system administrator can make sure that addresses correspond to the locally designated addresses for the AD. The packets coming into and going out of the AD should have the destination address and the source address assigned to the AD, respectively. The masking can be also applied to unavailable port services against portscan attacks if the network operator selects the NetViewer's port-based functions.

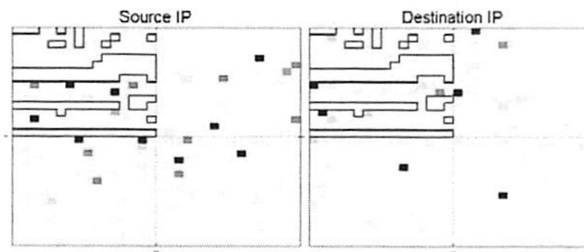


Figure 7: Spoofed source IP addresses.

Figure 7 shows the portion of NetViewer, where the blue-colored polygons indicate the reserved IP address space by IANA at the time of this writing. In legitimate traffic, there should be no pixels matching the unassigned space as shown in the destination IP image. Since packets can not be allowed to originate from or be destined to these spaces, the (colored) pixels matching the unallocated space have to be from spoofed IP addresses as shown in the source IP image. This attack is an instantaneous attack with (randomly) spoofed source addresses which aimed at a specific machine with 160 byte-sized packets.

The spoofed address masking can be expanded to IP bytes 1, 2 and 3 as well because the system operator knows the AD's internal usable address range (for stub networks). This also enables detection of sequential portscan attacks through the identification of addresses that do not provide service at the designated ports.

How Is NetViewer Different From IDS?

Network intrusion detection system (NIDS) is an important part of network security area and signature-based detection approach is being widely employed by network operators. The rule-based matching mechanism requires that the completed analysis of attack patterns and the availability of established remedies be available beforehand. To cope with new attacks, IDS tools are continuously required to be updated with the latest rules. Currently there are a few available freeware/shareware and commercial IDS tools.

We review Snort as representative IDS [ROE99], [SNO], and compare the properties of Snort and NetViewer. We perform this comparison by running the systems on a live, production network. We report

A1: Spreadsheet-based Scripting for Developing Web Tools

Eben M. Haber, Eser Kandogan, Allen Cypher, Paul P. Maglio, and Rob Barrett
– IBM Almaden Research Center

ABSTRACT

A1 is a Java-based spreadsheet environment that enables system administrators to build small tools that simplify and automate common tasks, integrating real-time data across heterogeneous systems. A1 spreadsheets can be saved to a central repository, where they are published and shared as interactive web portlets. In this paper, we discuss the need for administrators to create their own tools, how the A1 environment is designed to support this need, and how A1's support for web publishing – without requiring special web programming – can enable teams to share, modify, and improve their tools. We also discuss the design and implementation of A1, and show a number of sample spreadsheets for various administration tasks.

Introduction

System administration frequently involves the development of custom scripts to manage the unique requirements of each site. Typically, these scripts provide functionality lacking in commercial software tools because vendors cannot anticipate every task for every possible configuration in their tools. Custom-built scripts range from a few lines of code developed by a single administrator to speed a commonly executed task, to significant software systems developed by a team of people over several years.

Collaboration is a common aspect of system administration work. At most sites, expertise is distributed across teams who work together to design, implement, and maintain systems, and to troubleshoot problems. System administrators often share scripts as they collaborate on a task, or they pass scripts to others working on similar tasks. Scripts are often modified, customized, and improved, as they pass from person to person.

In the course of system administration field studies, we have observed several difficulties in existing scripting practices. Coordinated script use between different administrators can be hindered by a lack of central script repositories, inconsistent script versions, and varying execution environments. Web-based scripts provide consistent access and execution, but are difficult to develop. Many administrators lack the skills needed to field web applications, such as cgi scripting or J2EE servlet programming. Finally, we have seen many cases where the complexity of common scripting languages (e.g., Perl, Python, and shell scripts) prevented administrators from modifying and reusing existing scripts for their own purposes.

Though sometimes these issues result from limited programming experience, more frequently script sharing problems are due to insufficient documentation, hidden assumptions and hard-coded constants. All too often administrators who need a certain tool "re-invent the wheel," rewriting it themselves instead of building on existing scripts.

We developed A1 to enable system administrators to quickly develop custom web-based tools and to more easily share, reuse, and improve tools collaboratively. A1 is a spreadsheet-like environment that allows system administrators to invoke existing scripts or access remote systems via standard protocols, such as SSH, JMX and SNMP. A1 includes a task-specific scripting language so that connections to servers can be opened in a spreadsheet cell, and server actions can be triggered by changes to cell values. Sysadmins can save spreadsheets to a shared repository and then execute them from a web browser as interactive web portlets. A1 also provides a plug-in architecture to add support for new components.

A1 is not meant to replace existing scripting languages, such as Perl and Python. The power and flexibility of these languages will continue to be important for advanced users and experienced programmers creating complex tools. Rather, A1 is intended to provide an environment for a broader population of system administrators to create small tools that can be quickly and easily deployed and shared via the web. A1 spreadsheets can be also be used as a control layer to execute and manage the output of existing scripts developed in other languages. In summary, system administrators using A1 can build and share spreadsheets to access remote, heterogeneous systems, gather and integrate real-time data, and control various systems uniformly through a web-interface.

Background

Over the past three years, we conducted a series of field studies of system administration work. The study sites included large enterprise, university, and government research environments. We made fourteen

visits to six different sites, examining web hosting, database, operating system, security, and storage administration. We interviewed and surveyed system administrators about their work. We recorded a total of 50 days of video following normal, day-to-day activities. The results are primarily qualitative, providing a picture of typical work practices, tools, and problems faced by system administrators.

In these field studies, we made the following observations relevant to administrators' use of scripting and tools:

1) Collaboration is important for system administrators. Because no single individual can be an expert in every aspect of a large installation, responsibility is typically spread across people and organizations. During complex configuration or troubleshooting, collaboration is essential to a successful resolution. For example, in a detailed analysis of one lengthy troubleshooting session, we found that 90% of one system administrator's time was spent communicating with others, and that misunderstandings greatly delayed finding the solution [1]. In such an environment, there is great potential for improving effectiveness of system administrators by providing tools that allow them to better collaborate, communicate system status, and share control.
2) System administrators frequently create small custom tools to accomplish specific tasks. Administration tasks are complicated given the heterogeneous nature of most systems, with many components from different vendors and distinct local requirements. It is therefore not surprising that many tasks are not directly supported by vendor-supplied tools. We observed system administrators creating small tools on-the-fly to solve problems and collect information. Only one site we observed had a script repository, though the procedures regulating script publishing limited its use.
3) The programming abilities of system administrators vary tremendously. The system administrators we observed were very familiar with shell scripting, but general software and web development skills were usually not part of their job requirements. We observed the majority of scripts created by gurus. Novices would use scripts written by others, but they often did not feel comfortable modifying the scripts or creating their own. Other administrators between the novices and gurus would develop scripts occasionally, but often had trouble understanding and modifying scripts written by others. Administration work could be aided by easier-to-use and reuse scripting environments.
4) Administrators frequently integrate information gathered from different tools and different systems. This is especially true for heterogeneous systems containing components from different vendors in which each component requires a separate tool for access or configuration. Administrating heterogeneous environments could be improved by meta-tools that access and integrate information from many sources.

Successful tools and best practices often evolve and improve when custom tools are easy to create, modify, and share. Although web applications are readily accessible, only advanced users possess the skills to create them. Our goal with A1 is to provide an environment where most system administrators can create, share, and reuse custom web-deployable tools that integrate information from different systems. We want to leverage existing, common scripting abilities so that more administrators can create interactive web apps "for free," without having to become experts in cgi scripting or J2EE servlet programming. We also believe that shareable, web-based tools will be of significant value in aiding system administrators to collaborate.

Design

The primary challenges in creating a scripting language and environment for system administration are ease-of-learning to lower the barrier to entry for busy system administrators, providing broad solutions in heterogeneous environments, and enabling script sharing and reuse for teams of administrators.

To address these challenges, we took a spreadsheet-based approach. Over several decades, spreadsheets have proven to be easy-to-learn tools, particularly because of their straightforward programming style. Many system administrators already use spreadsheets as part of their regular work. The spreadsheet programming model encourages incremental coding, putting part of a calculation in one cell, using the output in a formula in another cell, and so on. Spreadsheets are also more robust than most scripting languages – the nature of spreadsheet execution means that errors only affect dependent cells, and do not necessarily invalidate the entire sheet.

The grid simplifies visual layout significantly, a common problem for less experienced programmers. Finally, the co-location of values and formulas in cells enables users to select a value and see the formula that defined it. This can help users understand and modify others people's sheets for their own use.

A1 extends traditional spreadsheets by (1) permitting cells to contain arbitrary Java objects, in addition to numbers, strings, etc., (2) extending cell formulas to include calls to methods of cell objects, and (3) allowing cells to contain procedural code blocks whose execution is triggered by events in the sheet. As with most changes that increase functionality, these extensions impact ease-of-use, but we believe the impact is small and the benefit large. Our use of Java takes advantage of a large base of existing libraries, particularly for IT management.

However, it is important to note that in A1 users do not need a deep understanding of Java. It is sufficient to know that objects have methods that can be called to change or return information about the state of the object. Procedural code blocks are triggered in a conceptually straightforward manner, either by changes to the value of specified cells or by cell formulas that evaluate to true. The conceptual model for execution is straightforward: "when something happens, perform these actions." The procedural code itself is written in a simple scripting language, with constructs for assignment, branching, looping, and cell method calls. Users familiar with shell scripting would find the A1 code language similar, and easy-to-learn. We also included traditional spreadsheet help features, such as pop-up menus on cell objects, so that users are not required to memorize all commands, functions, or methods – greatly improving A1's learn-ability through exploration.

A1 can be used to create broad solutions, tying together multiple scripts across different languages and systems. Within an A1 spreadsheet, each cell can invoke a different script and display its output, integrating the output of multiple tools or systems in one spreadsheet. Spreadsheets also bridge the gap between command-line scripts and graphical interfaces. With A1, users can push script output into charts and graphs, easily creating data visualizations of critical system status and performance. Data visualizations are currently under-used simply because they are too hard to create in most command-line environments, and vendor-supplied visualization tools are seldom sufficiently customizable.

However, A1 brings together useful qualities of both command-line (efficiency) and graphical interaction (ease-of-use). The spreadsheet layout also provides ample room for comments and explanation that can be co-located with command output. While all scripting languages permit diligent users to add their comments to help others understand script execution, such comments are often sparse due to programming under time pressure and the fact that comments do not improve script execution or usability; the beneficiary of comments is usually some one other than the script author. Adding comments to a spreadsheet immediately improves the sheet's usability because the comments are visible at execution time and can help clarify the sheet's operation.

To better enable script sharing and reuse, the A1 spreadsheet execution engine supports multiple rendering platforms. Spreadsheets are created using a stand-alone client-based tool with full interactivity, however the same spreadsheets can be saved to a central repository and executed without modification as J2EE-based portlets on a web server. We have integrated A1 with the ISC portal server, which IBM developed as a one-stop portal for system administration. ISC includes many vendor-provided portlets for

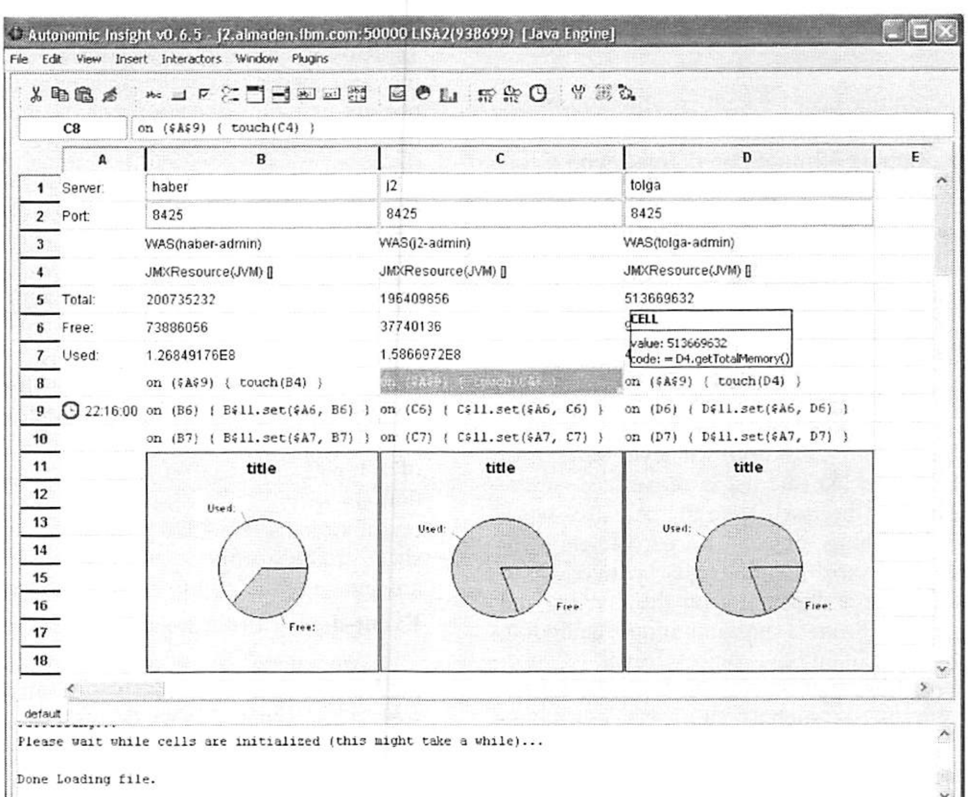


Figure 1: A JVM memory utilization monitor running in the A1 Client.

managing middleware products. Spreadsheets created by A1 will appear much the same as vendor-supplied system management portlets, with the additional option to edit existing tools or create new tools on-the-fly, extending the toolset with no installation and deployment requirements.

The A1 User Interface

As shown in Figure 1, the A1 interface appears very much like other spreadsheets, with a grid of cells, a cell expression editing field, toolbars and menus. Indeed, A1 supports all standard spreadsheet functionality: cells can contain strings, numbers, dates, and formulas that define a cell's value as a function of other cells. The primary difference is that A1 also supports Java objects as first-class cell contents, object-methods in cell expressions, and event-driven procedural code in cells. A1's enhancements to the spreadsheet language and execution engines enable users to easily build powerful tools that interact with live systems. A1 spreadsheets may also be deployed as web-based portlets. These features are described detail in the following sections.

Objects As First-Class Cell Contents

To support system administrators' needs for control of external systems, A1 adopted an object-based approach. Cells in A1 can contain any Java object, and cell formulas can refer to object methods. For example, if the cell A1 were defined as follows (in this paper, cell definitions will be shown with the cell name in brackets, followed by the contents of the cell):

```
[A1]  java.util.LinkedList()
```

then A1 would hold a Java LinkedList object from the java.util package. Other cells can refer to this object's methods, for example A2 might be defined with a formula such as:

```
[A2]  = A1.size()
```

so that A2 would always display a value indicating A1's linked list's size. References to objects and cell values can be mixed, for example cell A3 might be defined to use the LinkedList get() method, which returns the element at the specified index:

```
[A3]  = A1.get(A2-1)
```

This would cause cell A3 to hold whatever value is at the end of the linked list (the list is zero-based, so the (size-1)th element is the last). Note that A1 uses weak typing when matching objects to parameters. For example, numbers are automatically converted to strings and vice versa depending on the context and methods involved. From a programming standpoint, enabling cells to contain Java objects gives the user access to a large library of objects that provide support for programming needs, such as containers, I/O, statistics, data processing, and networking. From a system administration standpoint, users can immediately leverage a variety of existing administration APIs implemented in Java.

One important issue is how to render Java objects in a spreadsheet. In A1, each object class is associated with an *interactor* that specifies how it will be rendered on the screen and how the user may interact with it. Default interactors have been defined for most common object classes, though A1 allows users to change these defaults. Users can also define interactors for new object classes in their plug-in objects. A1 supports graphical rendering and interaction with objects. For example, a Boolean object can be rendered graphically as a CheckBox, or a NumberCollection can be rendered as an X-Y plot. If there are no interactors defined for an object, it is rendered textually using the default toString() method available in every Java class.

Interactors are designed to work on multiple platforms. In the current implementation, A1 supports three platforms: Java/Swing-based client, HTML-based portal server, and text-based command-line. Each interactor defines how to render and process events in each platform; for example, on the Java/Swing platform the interactor renders a Button and handles button presses in a standard GUI panel, and on the portal server platform the interactor automatically renders the Button as an HTML form input element.

For improved ease-of-use for those unfamiliar with Java, A1 provides a library of Java objects customized for spreadsheet use with a simplified set of methods and predefined interactors. These objects can represent rich data types (e.g., collections, queues, stacks), connections to external systems (e.g., SSH, SNMP, JMX), graphical widgets (e.g., Button, TextBox, ComboBox), and visualizations (e.g., X-Y plot, pie chart). Toolbar buttons exist for creating these objects, allowing the user, for example, to point and click to create an object that manages a server connection.

A1 includes a plug-in API for incorporating new Java objects into the framework. Experienced Java programmers can use this API to enhance existing objects with new interactors or with more sophisticated triggering mechanisms (e.g., pushing events into the spreadsheet, externally triggering spreadsheet re-evaluation). The API also allows customized objects to be added to the toolbar, permitting easier use of new objects in the spreadsheet. We believe the success of A1 will depend to a large degree on a rich set of domain-specific objects permitting access a wide variety of systems. We have provided a core set of spreadsheet friendly objects, but we hope that users take advantage of the API to create and share many more.

Event-driven Procedural and Functional Code

We found the strictly functional programming model of traditional spreadsheets insufficient in defining sophisticated control flows necessary for developing system administration tools. To enable richer control flow, A1 extends the spreadsheet language to permit cells to include event-driven procedural code blocks. These "micro-scripts" usually include a trigger indicating when

the procedural code block should be executed, either by a change in value for any of a list of cells, e.g.,

```
on (<cell address list>)
            { <procedural code block> }
```

or a boolean expression written in the same form as cell formulas, e.g.,

```
when (<boolean expression>)
            { <procedural code block> }
```

Continuing the example above, we might add code to clear the LinkedList contents automatically when its size exceeds 10 elements by putting the following in cell A4:

```
[A4]  when (A1.size() > 10){A1.clear()}
```

When the object in cell A1 changes, the size method is called. If the size is greater than 10, the linked list is emptied by the clear method. In general, code blocks can be triggered to execute upon changes to cell values, clock ticks, button presses, or any spreadsheet events. Procedural code blocks are written in a simple scripting language containing one or more semicolon delimited statements. A statement can call an object method (e.g., A1.clear()) or assign a new value to a cell (e.g., A10 = A3 * 10).

There is also support for conditional branching using an if statement, and iteration with a for statement (both of which work as in C or Java). When code blocks have no trigger expression, they must be explicitly executed from other code blocks by using the cell address as a statement (e.g., A4()). Code blocks may also force formula re-evaluation and code execution by using the touch statement (e.g., touch A1), which triggers all dependent cells as if the cell value (e.g., A1) has changed.

It is important to note that in A1 users can assign names to cells to improve readability of the code. For example, code in A5 can be named "insert":

```
[A5]  insert: {A1.add("10.0")}
```

and referenced in subsequent code blocks:

```
[A6]  when ( A1.size() < 5){ insert() }
```

For very small tools, using cell references is sufficiently clear. However, as tools grow in complexity, naming cells considerably aids readability and sharability.

Through these event mechanisms, users can achieve rich control flow in their programs. A spreadsheet cell can contain a button that, when pressed, triggers code to connect to a server and perform some action. Regularly polling a remote system can be implemented using a timer object that fires at specified intervals, with polling code triggered by the timer. Any object that uses the A1 plug-in API can also push external events into the spreadsheet, triggering reevaluation of formulas and execution of dependent code.

In addition to for loops, iteration is also supported in A1 through aggregate calls and recursive conditional constructs. For example, to reset a number of servers in cells C1 through C5, one would use the cell range notation ("..") in the method call, e.g., "(A1..C5).reset()". For conditionally bounded iteration, A1's event-based approach provides a simple solution through self triggering when() constructs, e.g.:

```
when (B1 < 10) {B1 = B1 + 1}
```

Web Portal-Based Collaboration Support

To support collaboration among system administrators, A1 spreadsheets can be deployed as portlets in a J2EE-based web portal server. In the web portal, administrators can execute or customize tools deployed by their colleagues from their web browsers. The web portal approach greatly helps shared situational awareness, as different sysadmins can see the same view of system status and controls.

All A1 spreadsheets are initially created in the Java Swing-based client application, which can be launched from within the web portal framework using Java WebStart. For example, Figure 2 shows the tool from Figure 1 running as a portlet. From the client tool, users can either save the spreadsheet locally (for their own use or further development), or to a server repository (for web deployment and sharing with others). Once in the server repository, spreadsheets may be run in a browser, like any other portlet. When a user wants to change a spreadsheet, they can launch the Java Swing-based client tool again using the "Edit" button on the portlet. When the sheet is updated, the user may either save the sheet locally, replace the existing portlet, or save it to the portal under another name. When a sheet is deployed to the portal and running in a web browser, objects are rendered and manipulated using the HTML form-based interactors. In this environment, spreadsheets are rendered to look no different than any other portlet application. To accomplish this, portlet cells that contain code are hidden from the user. Users can also explicitly hide other cells as appropriate.

Implementation Challenges

Implementing A1 involved several challenges, including building the underlying execution engine, and developing rendering and interaction techniques for both Java application and a J2EE-based environments.

The execution engine for a traditional spreadsheet relies on a dependency graph, recomputing each value when there is a change to dependent cells. Side effects are not permitted, and there are no guarantees as to the number of times that a formula might re-execute when triggered. Since A1 models real-world systems, we must handle side effects properly and ensure that code runs only when necessary. In addition, formulas might include method calls to remote systems that might not return results for an arbitrarily long time. To support these requirements, we implemented a custom multi-threaded execution engine.

One of the biggest challenges was the need to support vastly different interaction models such as the HTML form-based interaction and event-based Java/ Swing GUI interaction using a single language and

framework. In Swing applications, users can interact with GUI widgets that immediately generate events and cause objects to be rendered to reflect changes. In the HTML form model, input elements are contained in forms that are submitted and processed by a web server.

The difference is that in the web model users can make multiple changes on the form (such as entering text into multiple text fields) and submit the all the changes at once using a button on the form. This minimizes server round trips. A1 handles these two different models uniformly through the use of platform independent interactors and through "batched event propagation." Batched event propagation registers the current values of all "delayed" input elements (such as text input elements) on a portlet. Upon interaction with an element that causes a server request, all delayed input elements are processed first, before processing the element that initiated the request. Through this approach, A1 accommodates HTML form-based interaction much like the Swing GUI event model.

Sample A1 Tools

Simple System Monitor with SSH

This example demonstrates an A1 SSH object connecting to and executing commands on a remote server (Figure 3). The sample tool includes cells that contain labels and text fields for server name, login name, as well as a password field object used to avoid storing passwords in the script, and to hide them as users are entering passwords:

```
[B1]   "j2.almaden.ibm.com"
[B2]   "eben"
[B3]   PasswordField()
```

The SSH object is created in cell B5, referring to the values from the above cells:

```
[B5]   SSH(B1,B2,B3.getText())
```

Once the connection is successfully established, it can be used to execute commands and assign command output to a spreadsheet cell. Consider the example of a system where occasionally a problem causes the HTTP server processes increase in number. The sysadmin may want to occasionally check to see how many of these processes are running using the ps command. A button labeled "run" is placed in cell A6, with code in B6 that uses the SSH execute() method to run the command whenever the button is pushed, putting the results in cell C6 (Figure 4):

```
[A6]   Button("run")
[B6]   on (A6) {C6 = B5.execute
                 ("ps -ef | grep http")}
```

The example above can be extended to automatically restart the HTTP server and send email notifications. First, we replace the button with a clock object set to fire every 10 minutes, triggering cell B6 which updates cell C6 with the number of http processes. This, in turn, triggers B7, which checks if this number is

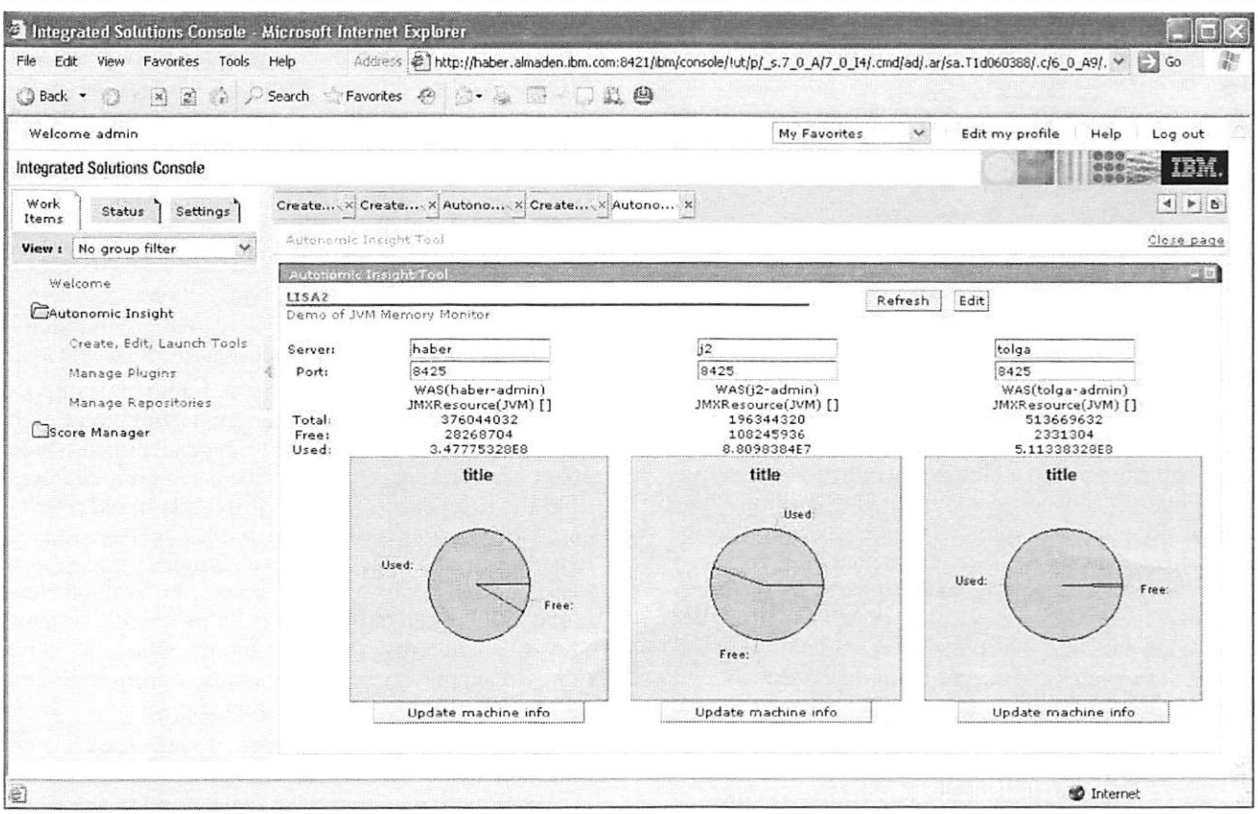


Figure 2: The JVM monitor running as a web portlet.

greater than 100. If so, it executes a command to restart the http daemon, putting any output in cell C7, and notifies the user by email by manually triggering the code in the cell named "notify" (a.k.a. B8).

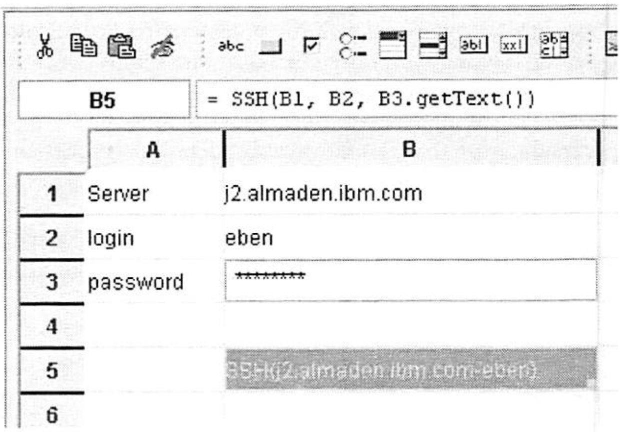


Figure 3: Creating an SSH connection.

```
[A6]   Clock(10*60*1000)
[B7]   when (C6 > 100) {C7= B5.execute
           ("/etc/init.d/httpd restart");
                                 notify()}
[A8]   MailService ("j2.almaden.ibm.com")
[B8]   notify: {A8.sendMail("root",
                 "eben","http server",
                 "server restarted"+c7)}
```

This example shows how A1 permits the user to quickly create a web-deployable tool that displays and controls the status of a remote system. This tool can be easily shared with other system administrators, who need only change the server, login, and password values to run the script within their own server environment.

Graphical JVM Memory Monitor

Figures 1 and 2 show a tool that uses Java Management Extensions (JMX) to connect to several WebSphere Application Servers (WAS) and display their Java Virtual Machine (JVM) memory usage using a pie chart. In this example, we show how to create that tool in detail.

First, we create labels in column A, and TextField objects in column B into which the user can enter the WebSphere server name and JMX listener port. The object name is visible when the user is typing. Once complete, the object is rendered, in this case as a fully interactive text field (see Figure 6).

Next, in cell B3 we create an object that encapsulates a JMX connection to the WebSphere server:

```
[B3]  WAS(B1,B2,"","")
```

The code creating the WAS object refers to the values that the user types in to cells B1 and B2, and uses empty values for login and password (as authentication for JMX is turned off). In JMX, servers can return one or more MBeans, which contain a number of attributes and methods. To retrieve an MBean from a WAS JMX object, we will call the getResource() method in B4, specifically asking for the JVM MBean:

```
[B4]   = B3.getResource("JVM")
```

If users can't remember all methods available on an object, it is possible to right-click on the object to pop-up a method list (Figure 6). Once we have the JVM MBean, we can get further information by using methods such as getTotalMemory() and getFreeMemory() to get the memory statistics into cells B5 and B6, respectively:

```
[B5]   = B4.getTotalMemory()
[B6]   = B4.getFreeMemory()
```

It turns out that the JVM MBean doesn't have a method for returning used memory, but that is easily computed from the difference between total and free using a regular spreadsheet expression in cell B7:

```
[B7]   = B5 - B6
```

Now we have cells that show total, free, and used memory. Because this particular API does not support "pushing" data from the server, we need to regularly poll the server to keep the values up to date. This is

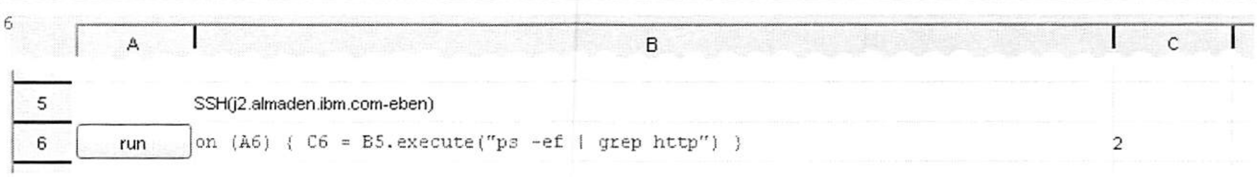


Figure 4: Running an SSH command triggered with a button.

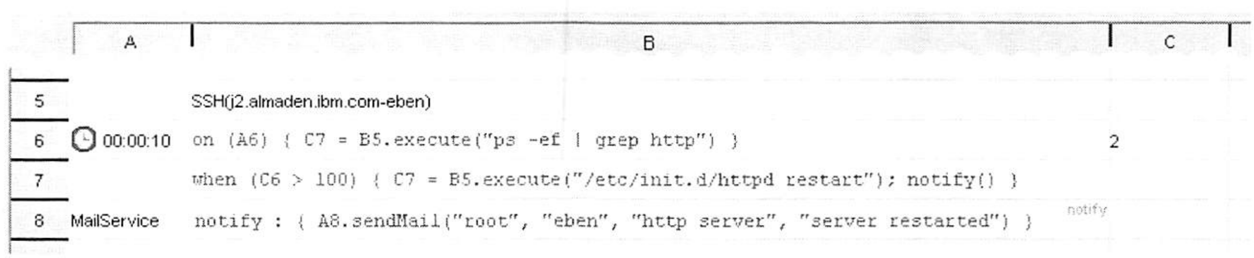


Figure 5: Automating HTTP server restart.

accomplished with a timer and code to trigger updates. Cell A9 holds the Clock object,

```
[A9]   Clock()
```

and cell B8 contains code that, whenever the clock fires, "touches" cell B4, causing all dependent cells to be reevaluated:

```
[B8]   on ($A$9) { touch(B4) }
```

Note here that we used the "$" modifier in the cell reference to make it an absolute cell address, thus ensuring that when cell is copied to another location, the expression will still refer to cell A9. This will make it easier to duplicate the code for use with other servers. Now the memory values are being updated at regular intervals. The display can be improved, however, since textual number displays aren't the best way for displaying relative values such as used and free memory. We now create a pie chart object in cell B11:

```
[B11] PieChart()
```

To populate the pie chart with values, we add code in cell B9 to have a pie slice with a name from A6 ("Free") and value from B6:

```
[B9]   on (B6) { B$11.set($A6, B6) }
```

Thus, whenever the value in B6 changes, the pie slice is updated. In the reference to B$11, only one "$" is used in the cell reference, making the row of the chart invariant. This permits us copy/paste cell B9 to B10, where it will create a pie slice for "Used" memory, and then copy both cells to other columns for other servers. The reference to $A6 makes the column invariant, since the labels are only found in the first column. We now have a complete memory monitor for a single server. Expanding it to support more than one machine is relatively easy. We just select cells B1 through B11, copy and paste them into columns C and D. A1 supports standard spreadsheet cell expression rewriting, so non-absolute cell references in formulas are changed when pasted to refer to the new columns.

The working memory monitor is shown in Figure 1. The memory monitor can also be saved to a server repository where it becomes immediately available as a portlet tool. Figure 2 shows a snapshot of the portlet tool as it appears running in a web browser.

Creating a Server Inventory Report

We have observed system administrators receiving management requests for inventory reports of all servers owned by an organization. The requests came from different people, and involved different groups of servers. The requested information included OS type, version, platform, hardware details (processor type/speed, memory), storage utilization, network interfaces, machine location, etc. The sites involved had various versions of Windows, UNIX, Linux, and OS/2, so capturing the required information required a variety of mechanisms, e.g., SSH, SNMP, management tools, and databases. The system administrators we observed collected such information manually, entered into a spreadsheet, created necessary charts, and sent the report to management. Given the nature of this task, it seems ideal to use A1 to both retrieve and organize the data.

Here, we show two examples of such data collection, one in Windows through SNMP and the other in Linux through SSH. Each machine is listed on a separate line, so system administrators can easily duplicate it for as many servers as necessary through copy and paste. From these examples, it should be clear how other server and OS types and access methods could be implemented.

First, we create a set of buttons to trigger updates on system information (Figure 7). In cell A1, we put a button to update all server information, and each line corresponding to a server also has an individual button to specifically update that server's information.

```
[A1]          Button("Update All")
[B4,B5,...]   Button("Update")
```

Cells C4, C5, and so on contain code to trigger that row's button whenever the master button in A1 is hit. For example, C4 will contain:

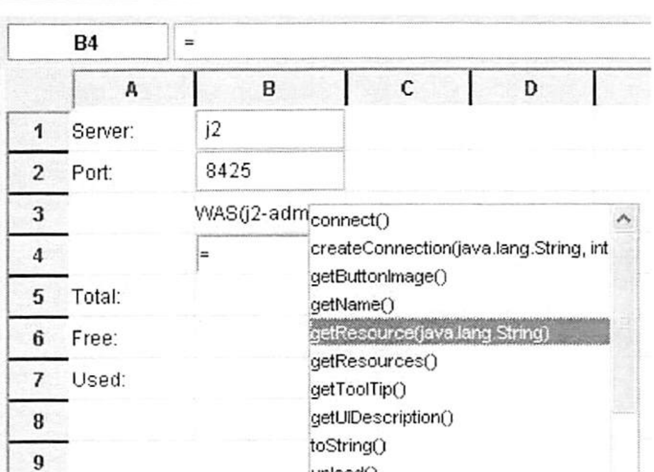


Figure 6: Using a popup menu to list methods.

```
[C4]   on ($A$1) {touch(B4)}
```

For each machine, column A will be the machine type/access method for the convenience of the spreadsheet user copying and pasting rows (e.g., Windows/SNMP, Linux/SSH), and Column D has approach specific code to connect to remote server. For SNMP and SSH the code is:

```
[D4]   on (B4) { E4 = SNMP(I4, "", "") }
[D5]   on (B5)
       { E5 = SSH(I5, G5, H5.getText())}
```

Whenever the button is pressed, a connection object is created, causing other values in the row to be updated. Connection status is reported in the fifth column with a simple method call:

```
[F4]   = E4.testConnection()
```

The subsequent three columns include login/password (if needed), and host name. The following columns contain formulas that derive machine information from the connection object. With SNMP, we get the machine description, uptime, #network interfaces, and network interface descriptions as shown below, respectively:

```
= E4.getResourceValue
                  ("1.3.6.1.2.1.1.1.0")
= E4.getResourceValue
                  ("1.3.6.1.2.1.1.3.0")
= E4.getResourceValue
                  ("1.3.6.1.2.1.2.1.0")
= E4.getResourceValues
              ("1.3.6.1.2.1.2.2.1.2.*")
```

The same information can be retrieved from a Linux machine via the following use of the SSH object:

```
= E5.execute("uname -osrv")
= E5.execute("uptime")
= E5.execute("netstat -i | wc -l") - 2
= E5.execute("netstat -i")
```

Other pieces of system information may be extracted in a similar fashion, though some require more extensive textual processing, either using shell commands through SSH, or using one of the built-in string processing functions provided by A1. Though at first it may seem to be complex, once a row collecting information for one machine is completed, it one can be copied and pasted for all other machines of the same type.

Evaluation

A1 is currently an alpha release. We completed one laboratory usability study and are currently conducting field trials. Our laboratory study involved twelve participants, seven professional sysadmins and five programmers. Of the seven system administrators, two had more than four years scripting experience, and the remaining had zero to two years experience. One goal was to determine whether A1 could be learned and successfully used in a two-to-three hour period.

In the study, participants first reviewed a self-paced tutorial of the A1 programming language and user interface. Second, participants practiced using A1 by developing a simple tool following explicit step-by-step instructions. The first two steps typically took about 30 minutes each. Next, participants were asked to develop two system administration tools, one based on the other, given only descriptions of the tool requirements. Finally, an interview was conducted with participants to learn about their experiences.

The first tool to be created was a log-space-monitoring and backup-automation script for an http server. The tasks included querying the http server for the name of its current log disk, using that name to query the file system for log disk utilization, and registering listeners to monitor disk utilization. When the log disk is nearly full, code must notify the system administrator through email, and when the log fills completely, it must stop the http server and start a disk backup. The second tool extended the first one through code to switch log disks automatically when two log disks are available.

After taking the tutorial and building the sample tool, the participants spent between one and two hours working on the subsequent tools. Overall, they were fairly successful, on average completing 80% of the required functionality. Given the complexity of the tools and the limited time to learn the language and environment, these results are very encouraging. Equally encouraging were positive comments by the participants on the incremental and interpreted nature of A1. One said, "It allows me to do things in my own order. I can refer to a non-existing object. I know I'm going to create the object next." Another said, "The fact that I can interact with my systems in real time in the spreadsheet that alone is pretty cool." Another added that building tools this way was "the quickest live application [I] ever made."

The results did highlight several areas for improvement. Learning the object-oriented programming model proved somewhat of a hurdle for the less experienced participants. In addition, many participants had difficulty understanding the semantics of the objects and methods used in the tools. These problems

	A	B	C	D	E	F	G	H	I
1	Update All								
2									
3	Approach	Update	Code 1	Code 2	Access Object	Status	Login	Password	Host
4	Windows/SNMP	Update	on (A1) { touch(B4) }	on (B4) { E4 = SNMP(I4, "", "") }	SNMP(habert23-)	OK!	<na>	<na>	habert23
5	Linux/SSH	Update	on (A1) { touch(B5) }	on (B5) { E5 = SSH(I5, G5, H5.getText(	SSH(j2-eben)	OK!	eben	********	j2

Figure 7: An inventory spreadsheet.

suggest a need for improved introductory documentation to explain the programming model, and better interactive browsing of object APIs to aid programmers in learning object and method semantics.

We are also conducting field trials with system administrators, both inside and outside our company, to see if they can develop tools that are useful in their work settings. So far, A1 has been field-tested in three different groups: web-hosting, server management, and cluster management. The study primarily included requirements gathering, use-case and template development, first-hand experience by system administrators in tool building, and data collection regarding problems and issues with A1 user interface, language, environment, and libraries. In these, A1 was used for developing tools for recycling a steady-state server, creating system inventory reports, and monitoring cluster server performance.

Overall, the reaction to A1 was quite positive. Users especially liked live connections to systems, easy integration of data, and the web interface. One of the users said: "... but this is like you have real time technical data, and accurate." Integration was a major concern among users in their daily work. One user said: "... this is the only tool that brings all this data together." Another referred to the difficulty of creating web interfaces in one of their projects: "[to get a] web front in front of it (the report) it took six months and it still didn't work right," adding "[with A1] it was on the fly, it is great!" Users also appreciated the fact that A1 does not require installing agents on servers, pulling data from systems. Issues concerning A1 including lack of support for running scripts as background jobs, need for more interactive scripting and improved text processing utilities, and lack of sufficient documentation and online help on object libraries.

Related Work

Scripting is pervasive in system administration, primarily through command-line shells. System administrators often open a number of shell consoles, execute command-line instructions and automate tasks through languages such as Perl, Python, and C Shell [10]. System management tools are typically vendor-specific-each vendor provides a management tool for its own systems, such as IBM's DB2 Control Center. Tivoli and EMC, two leading providers of tools for system administrators, have hundreds of tools for enterprise-wide operations. To address the fact that system management activities often span multiple tools from multiple vendors, both Tivoli and EMC have integrated environments. For instance, Tivoli's Enterprise Console provides comprehensive management capabilities, including monitoring systems from multiple vendors.

One limitation of these environments is that they provide little support for customization. For example, the systems typically offer standard, prepackaged charting capabilities without a means for end users to create their own system views. PIKT [13] and Cfengine [3] are well known in the system administration community as domain-specific languages for monitoring and configuration management of diverse environments. Unfortunately, complexity of these languages and lack of supporting visual environments make them suitable mainly for experts.

Related work on spreadsheets is extensive in both the research and commercial realms. Spreadsheets were one of the original "killer apps" that prompted wide use of personal computers. While effective for tabular calculations, traditional spreadsheets lack expressibility and programming power [17, 4] and they are limited in their ability to interact with external systems. Researchers have taken a variety of approaches to these problems, investigating changes to spreadsheet language, programming model, data types, and user interface.

Of particular relevance is work on object-oriented spreadsheets [2, 5, 9]. These approaches use object models specific to their systems, and require new languages for object definition. Our use of Java has the advantage of a large base of existing users, tools, and libraries, particularly for IT management. In addition, the other approaches were purely functional programming environments, whereas we think our procedural code blocks provide a more natural means of managing the side effects (such as restarting a system) inherent in system administration.

Commercial spreadsheet systems (e.g., Microsoft Excel) address limitations in programming power and access to external systems by permitting users to write programs in an external language (e.g., in Visual Basic). This approach has several drawbacks. First, data sources and processes are not explicitly represented in the spreadsheets as first-class cell contents (unlike numbers, strings, etc.). Thus, interaction with external processes is done through a language and programming model (e.g., Visual Basic) different from that used in the spreadsheet and typically requires different programming skills. Second, functionality provided by external processes (e.g., operations defined on servers, such as *shutdown*) may not be exposed to the user in a form that can be used readily in spreadsheet expressions. Finally, data are typically pulled from external processes rather than actively pushed by these processes (though in Visual Basic it is possible to program this explicitly too). These issues are all important for system administration.

A1 extends previous work through explicit representation of external systems as objects in cells. These objects expose their properties and methods to the user, who can use them in functional expressions or procedural code blocks to query and control systems. Unlike previous approaches, A1 carefully combines procedural and functional constructs through an event-based approach to

achieve rich control structures for the system administration domain. A1 takes the inherently event-based programming of spreadsheets to the extreme, where each cell can either explicitly or implicitly create, listen, combine, and process events. Most importantly, A1's extensions are carefully crafted to create a model and language that conforms to the spreadsheet metaphor while remaining usable, and powerful.

Discussion and Future Work

A1 is a working prototype that aims to support system administrator tool building. There remain open questions as to how well it will work in the real world. We are currently involved in field trials to answer specific questions:

- Is the A1 environment sufficiently easy to use (and reuse) to permit a broader population of system administrators to do their own scripting?
- Is the A1 environment sufficiently powerful and usable to be useful for real system administration tasks?
- Can A1 be learned quickly enough so that administrators are willing to make the time investment to learn it?
- How valuable is a web portal repository of shared tools?
- Does A1 have any scaling limitations for real world tasks?
- What set of access methods (SSH, SNMP, JMX, etc.) is needed?

We hope to better understand how to make A1 successful in the real world through our field trials, and through a public release of A1. There are several areas in which we are already working to improve A1. Error handling and debugging of scripts need improvement, since our extensions to the programming model can make dependencies harder to understand. In addition, right now A1 has no mechanism for logging spreadsheet events, a necessary feature when using A1 to manage systems. We are also actively seeking to develop more plug-ins for connecting to a wider variety of systems through additional protocols.

Conclusion

We have developed a tool-building environment to support system administrators in working collaboratively to create custom solutions for their site. It provides access to a wider audience by combining a spreadsheet-style development environment with web portal deployment. An early version has been made available on the web, and we are actively testing and developing it to make A1 a useful tool for system administrators.

Availability

A1 is scheduled to be available on IBM alphaWorks in October 2005.

Acknowledgments

We thank the system administrators who participated in our field studies and in the preliminary study of A1 for their time and their thoughtful suggestions.

Author Information

Eben Haber works on Human-Computer Interaction at IBM Almaden Research Center. He holds a Ph.D. from the University of Wisconsin-Madison, where he worked on improving user interfaces for database systems. His interests include databases, user interfaces, and the visualization of structured information. He has worked in industry on data mining and visualization, user interface design, and is currently studying human interaction with complex systems in the USER group at IBM Almaden.

Eser Kandogan is a research staff member at IBM Almaden Research Center. He holds a Ph.D. from the University of Maryland, College Park, where he studied computer science with a specialization on Human-Computer Interaction. His current interests include human interaction with complex systems, policy-based system management, ethnographic studies of system administrators, information visualization, and end-user programming.

Allen Cypher is a research staff member at IBM Almaden Research Center. He received a Ph.D. in Computer Science from Yale University. His research interests are in end-user programming, and he has worked in industry designing user interfaces, programming environments and custom tools. Paul Maglio is Senior Manager of Human Systems Research at the IBM Almaden Research Center. In his nine years at IBM Research, Paul has worked and published extensively in the areas of human-computer interaction, intelligent agents, web intermediaries, system management, and autonomic computing. He has a Ph.D. in cognitive science from UCSD, and an SB in computer science and engineering from MIT.

Rob Barrett is a Research Staff Member at the IBM Almaden Research Center in California where he works in the Services Research group on bringing value from human-computer interaction research to the IBM Global Services organization. His current work focuses on the user experience of system administration and human aspects of autonomic computing. Previous work includes an intermediary approach to designing web applications, optimization of pointing devices, track-following servo systems for tape data storage, and atomic-scale imaging. He holds a Ph.D. in Applied Physics from Stanford University and has earned masters and bachelors degrees in physics, electrical engineering and theology.

References

[1] Barrett, R., E. Kandogan, P. P. Maglio, E. M. Haber, L. A. Takayama, M. Prabaker, "Field Studies of Computer System Administrators:

Analysis of System Management Tools and Practices," *Proc. CSCW*, 2004.

[2] Burnett, M., J. Atwood, R. W. Djang, J. Reichwein, H. Gottfried, S. Yang, "Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm," *Journal of Functional Programming*, Vol. 11, Num. 2, pp. 155-206, March 2001.

[3] Burgess, M., "A Site Configuration Engine," *Computing Systems*, Vol. 8, Num. 1, p. 309, MIT Press, Cambridge, MA, Winter, 1995.

[4] Casimir, R., "Real Programmers Don't Use Spreadsheets," *ACM SIGPLAN Notices*, 27, pp. 10-16, June, 1992.

[5] Clack, C., L. Braine, "Object-oriented functional spread-sheets," *Proc. 10th Glasgow Workshop on Functional Programming (GlaFP'97)*, September, 1997.

[6] Couch, A., "An Expectant Chat About Script Maturity," *Proceedings of LISA 2000*, pp. 15-28, 2000.

[7] Gittler, X., K. Beer, "Designing a Configuration Monitoring and Reporting Environment," *Proceedings of LISA '03*, pp. 61-72, 2003.

[8] Hagenmark, B., K. Zadeck, "Site: A Language and System for Configuring Many Computers as One Computer Site," *Proceedings of the Workshop on Large Installation Systems Administration III*, p. 1, USENIX Association, Berkeley, CA, 1989.

[9] Hudson, S., "User Interface Specification Using an Enhanced Spreadsheet Model," *ACM Transactions On Graphics*, 209-239, July, 1994.

[10] Joy, William, *An introduction to C Shell.*

[11] Libes, D., "How to Avoid Learning Expect -or- Automating Interactive Programs," *Proceedings of LISA '96*, 1996.

[12] Myers, B. A., J. F. Pane, A. Ko, "Natural Programming Languages and Environments," *Communications of the ACM*, Vol. 47, pp. 47-52, September, 2004.

[13] Osterlund, R., "PIKT: Problem Informant/Killer Tool," *Proceedings of LISA '00*, pp. 147-165, 2000.

[14] Pierce, C., "The Igor System Administration Tool," *Proceedings of LISA '96*, 1996.

[15] Stepleton, T. "Work-Augmented Laziness with the Los Task Request System," *Proceedings of LISA '02*, pp. 1-12, 2002.

[16] Wack, A., *Partitioning Dependency Graphs for Concurrent Execution: A Parallel Spreadsheet on a Realistically Modeled Message Passing Environment*, Ph.D. Thesis, Department of Computer and Information Sciences, University of Delaware, 1995.

[17] Yoder, A. G., D. L. Cohn, "Real spreadsheets for real programmers," *Proc. ICCL '94*, IEEE Press, pp. 20-30, 1994.

HostDB: The Best Damn host2DNS/DHCP Script Ever Written

Thomas Limoncelli – Cibernet Corp.

ABSTRACT

HostDB is a system for generating DNS zone files, BIND configurations, and ISC DHCP server configurations. It is extremely simple yet powerful, which makes it easy to deploy. It is not so complicated that it requires an SQL database or web server. It is not a single generation script but a complete system that includes tools for deploying the files that are generated. The system is written in Perl and shell and makes heavy use of make. The zone files that it generates look hand-made, except that they are clean. Powerful tools are also provided to aid in the conversion from legacy systems. HostDB is the third generation of my DNS maintenance systems and embodies not just superior software but also best practices in system administration. HostDB uses small tools that combine to achieve large goals. Initial deployment can be done in stages to reduce risk. Deployment of updates is fully automated. It rocks.

Introduction

The goal of HostDB is to create a DNS zone generator that is full-featured without requiring complicated databases and web services. The system uses simple configuration files and automates deployment. It is easy to deploy whether you are converting a legacy system one zone at a time or setting up a new system from scratch.

Many small and medium sites (dozens or hundreds of machines) hand-edit DNS zone files. This is an error-prone practice which requires a high skill level. The smallest typo can cause big outages.

Many large sites (multiple IT teams, locations, and internet connections) use DNS management systems that provide no revision control and very little audit trail of who made what changes. Many also require external (outside the firewall, or "public") zone files to be maintained by hand, which adds the risk that they will become out of date with internal (inside the firewall, or "private") zones.

The problem with most DNS generation scripts is that they are either too simplistic to be useful or so complicated that they are intimidating to install. The simple ones are fine for doing an initial conversion from /etc/hosts to zone files that will be hand-edited in the future. Many are "point tools" that require a lot of integration work. The complex ones are so unwieldy that small sites find them too scary to install. Most don't handle complicated DNS idioms such as having special MX records for mail servers, proper handling of multi-homed hosts, or templates for ranges of similarly named hosts. They don't handle generating different zone files for inside the firewall versus outside the firewall (i.e., "host hiding") resulting in hand-edited external zone files that are prone to errors. Many generate zone files that are not pretty: hand-edited zone files would be more readable (if we only had the time to maintain them that way!). HostDB fixes all these issues. It does this without introducing a complicated syntax plus it generates your DHCP configuration file "for free"! The code is written in Perl, shell, and make and is freely available.

Setup time is fast. Really fast. The system can be used to generate zone files that can be individually inserted into your legacy systems within minutes of installing the software. HostDB breaks from the Free and Open Source Software (F/OSS) tradition of providing a default configuration that is an arrogant display of the most complicated features. Instead the default configuration is a useful example that can often by used "out of the box" with only small changes. Documentation is included for migrating from a legacy system in small steps with plenty of testing along the way.

Audience

HostDB seeks to fill the void between simple DNS zone generation scripts and massive DNS/DHCP database-driven applications. It is perfect for a site that is starting fresh, or for one that currently hand-edits zone files and wishes to introduce automation.

The learning curve and installation curve for HostDB is very small. Thus, it is appropriate for sites that do not have the need, expertise, or resources required by systems that have a web-based front-end or use a SQL database as the backend.

HostDB is good for sysadmins that need an extremely good solution but don't have a lot of time. Migration from a legacy system is extremely easy because the files that are generated are put into a

repository before they are copied to their final destination. Thus, comparisons between the legacy zones and the HostDB-generated zones can be done as part of acceptance testing. Tools for comparing zones are provided. Individual zone files can be put into service permitting an incremental deployment methodology (rather than a "flash-cut"), thus reducing risk.

HostDB is not appropriate for extremely large organizations with many separate IT groups or sites with complicated dynamic DNS configurations. Their needs would be better served with other, larger, systems. It doesn't directly support a sophisticated authorization schemes, a web-based interface, or assistance for dynamic DNS updates. However, HostDB does have some hooks that make it easy for simple dynamic DNS to be used. Also, large organizations are often composed of many smaller IT organizations which would find HostDB entirely appropriate. These organizations often have one team that manages all internet connectivity and HostDB is excellent for maintaining the related DNS zones.

At this time HostDB has no support for IPv6. I'm okay with this.

What Does It Do?

I have seen many exceptional DNS/DHCP configuration management systems that provide excellent web front ends and are so powerful that implementation requires an SQL database. HostDB isn't one of those.

HostDB lets you track all DNS and DHCP information about all your hosts in a single, simple file. From that file, it generates DNS zones, BIND configuration files (named.conf) and host entries that can be included in an ISC DHCP server configuration (dhcpd.conf). Two helper configuration files guide zone creation and file deployment, but they are basically static after the system is stable.

The system generates the complete zones and configurations. No hand-editing after the fact is required. Thus, automation using make and other tools is possible.

The system handles complicated DNS idioms such as:

- Special MX records for mail servers.
- Proper DNS records for multi-homed hosts and routers.
- Templates for ranges of similarly named hosts.
- Host-hiding (generating different zones for external DNS servers).
- Clean, human-readable zone files are generated (as if they were human-generated, but neater)

HostDB includes a tool called "mkdestinations" which generates a Makefile that will efficiently deploy (push) individual files to local and remote hosts. It recognizes zone files and treats them specially to prevent zone-transfer storms. Non-zone files can be deployed as well, making it possible to maintain the all files related to DNS and DHCP configuration for multiple servers around a network from one central administration host.

Major tools are written in Perl, with many helper programs in BASH shell. Shell glues them together in a way that is easy for sysadmins to understand. The system tries not to re-invent the wheel, thus it uses standard UNIX tools: make, SSH, sed and awk. Sed and awk have been feeling ignored lately so we threw some work their way.

How to Configure

There are three configuration files. Each encapsulates information that would be updated by a particular audience. The format is either simple or complicated based on the audience as summarized in Table 1.

The syntax of hostdb.txt is simple because a non-DNS expert needs to be able to edit it without introducing errors. Rather than designing a syntax by and for DNS experts, much effort was spent in designing a syntax that made sense to the junior engineers that would be using the system. The filename ends with ".txt" to facilitate non-UNIX admins editing these files through non-UNIX tools like Notepad.

zoneconf.txt requires a little more knowledge of how DNS zones work, which is appropriate because this file should only be edited by a person with advanced DNS knowledge.

destinations.txt is extremely simple because it simply states where files are to be copied after they are generated. It permits files to be renamed as they are copied, which makes inserting a generated file into a pre-existing legacy system very easy.

Standard UNIX protections can be used to control who can edit these files. They are all plain ASCII files, appropriate for maintaining under revision control systems like RCS, SCCS, CVS or Subversion. Any system administrator maintaining key files such as these without a revision control system is taking an undue risk. Revision control means knowing what changed when. When a change is introduced that causes an outage, one can undo a change, investigate who made the change, mentor them so they do not repeat the mistake, and optionally ridicule them during staff meetings.

DNS Zone Generation

DNS zones are generated using "mkzones". Mkzones parses the entire hostdb.txt file, storing the important bits into various data structures. It then reads zoneconf.txt which is "executed" line by line, setting options and generating zones as instructed. Because of this design, an option can be set one way before a particular zone is generated and then another way before the next zone is generated. This is a powerful feature, letting one system generate zones with very different

needs. For example, if one sub-domain requires different default MX records than all the others, it is very simple to achieve that goal. Other DNS generation systems assume the same options for all zones. That makes them inappropriate for very large sites.

The power of mkzones' algorithms comes from the fact that the parsing of hostdb.txt is decoupled from the generation of the zones. This permits more sophisticated zone generation since the generation algorithms understands "the big picture," i.e., all host information is available at once.

Decoupling parsing from generation also had the benefit that the input syntax could be changed without affecting the generation code. We were able to experiment with different syntaxes over time without worrying about how this would break the generation code.

The syntax for hostdb.txt was developed with massive user input. Users were presented with options for syntax and their feedback altered the evolution of the system. Most DNS zone generation systems tend to be designed for the DNS expert. HostDB has the benefit of a host list which is in a format that is perfect for junior sysadmins and the complicated options are kept far away in a different file where only the experts can make changes.

The users quickly lined up behind a syntax that looked like /etc/hosts, not a fancy recursive syntax like ISC's BIND and DHCP configuration files. They requested a single line per IP address because when it is time to allocate an IP address (their most frequent task) the procedure they are most comfortable with is finding an unused IP address in a long list, sorted by IP address. They also didn't want to be bothered with having to keep the file perfectly indented or brackets matched.

What Makes the Generation So Cool?

To make a zone file look readable took quite a long time. After many iterations we finally realized that the most readable zone file stripped hostnames (removed the domain, when possible), were formatted with tabs (not spaces), grouped records related to a particular host, and ordered the hosts by IP address.

Therefore, when generating zone files, we output:

1. The SOA information
2. The NS records
3. Any records for the current zone name (the domain without any host)
4. All the DNS information related to a particular host (with hosts listed in IP address order)

You may notice that the examples from hostdb.txt always specify Fully Qualified Domain Names (FQDNs). That is, foo.example.com, not just simply "foo". Usually DNS systems assume a label is simply a hostname (sans domain) unless it ends with a single period. However, humans tend to forget the period. After many iterations of trying to solve this problem by being fancy, we decided it was better to require the users to always enter FQDNs and output warnings if something was not a FQDN.

The decision to group the records in the zone file by hostname (yet stay in numerical IP address order) was something of a small breakthrough discovered after many trials. To do this all DNS information has to be in memory before the first zone is generated. This way decisions can be made by checking in-memory data structures and flags that were built up during parsing. Items from the in-memory data structures are deleted as each DNS resource-record is output into the zone file, thus eliminating the possibility of outputting the same line twice.

BIND Configuration File Generation

mknamedconf uses the information in zoneconf.txt to generate ISC BIND configuration files for DNS masters and slaves inside the firewall, plus DNS masters outside the firewall. It does not generate configurations for slaves outside the firewall (external) because the author has not yet found the need for such a thing. External slaves tend to be owned and managed by separate organizations, often ISPs. We do not know enough to generate their entire bind configuration, nor can we assume they use BIND. Configuration updates are usually done via manual (emailed) requests of the owner of the slave server.

DHCP Generation

The DHCP generation is very simple. Mkzones is called with a flag that tells it to parse the hostdb.txt file as usual, but then output a ISC DHCP "host" statement for each machine with DHCP-related data. This information can be included in a ISC DHCP "dhcpd.conf" file, which must be manually created. DHCP configurations are usually too custom to warrant generating them automatically, except for the host statements.

File Name	Audience	Contents	Skills required
hostdb.txt	Anyone on your sysadmin team	Info about all hosts	Edit text files, maintain a format set up for them, understand IP addresses
zoneconf.txt	DNS engineers	How the zone files and BIND configuration files are generated	Understanding of DNS zones and terminology
destinations.txt	UNIX admins setting up BIND	Where zone files and other files are copied	Knows where zone files are stored and has root access on DNS servers

Table 1: HostDB configuration files.

The Syntax

The systems works in two phases. First, the zoneconf.txt and hostdb.txt files are used to generate all the files that are generated. Next the new files are copied to the appropriate hosts/directories based on a map described in destinations.txt.

You might think of this as:
Hostdb.txt + zoneconf.txt → generated files
Generated files + destinations.txt → deployed files.

hostdb.txt Syntax

The hostdb.txt file looks like a /etc/hosts file augmented with flags and parameters. Display 1 demonstrates the syntax of hostdb.txt:

The zone file "example.com" that would be generated would be a simple "A" record for each host plus appropriate NS and MX records as configured in zoneconf.txt. The reverse zone (1.1.10.in-addr.arpa) would be one PTR file as appropriate for each host.

Notice that each host is listed by its fully qualified domain name (FQDN). For example, spoon.example.com instead of just "spoon". During user testing, it was discovered that it was too difficult to come up with a syntax that permitted "short" hostnames to be used and still permit multiple domains to be described in the same file. Various formats were attempted. In the end, it was decided that it was cleaner to make every host be listed as a FQDN.

"kettle.example.com" has a special option on it: "MAC=00:b0:d0:a6:cf:f1". This identifies its Media Access Control (MAC) address, or Ethernet, address. When a DHCP configuration file is generated, this host will have a "static assignment" or "permanent lease" for 10.1.1.2. We have found it useful to assign static assignments to all known hosts, and use "pools" of randomly allocated addresses only for transient hosts. This makes log files more accurate since most machines will always appear at the same IP address.

"fork.example.com" has a CNAME of "pitchfork.example.com". This generates a DNS CNAME record.

In addition to CNAME=, there is also ANAME=. This generates multiple DNS A records pointing to the same IP address. HostDB generates the reverse lookup to be the first host on the line, which seems to be the most common practice.

Multiple ANAMEs and CNAMEs can be listed by separating them by colons. For example one might list "ANAME=pitchfork.example.com:branch.example.com:divide.example.com".

ANAMEs and CNAMEs can both appear on the same host line.

The plural ANAMES is the same as ANAME, as is CNAMES the same as CNAME. This makes the hostdb.txt file slightly more readable.

Multihomed Hosts

HostDB understands that multihomed hosts need special treatment. For example, a router has many interfaces. The best practice is to have one DNS label or name that will return all "A" records; one for each interface. However network engineers and others need to be able to specify a particular interface when they want. HostDB likes it both ways.

Suppose the .1 address of each network was an interface for our router, zathras. We could specify that as shown in Display 2.

The DNS zone information generated would be:

```
zathras         IN  A    10.1.10.1
                IN  A    10.1.20.1
                IN  A    10.1.30.1
                IN  A    10.1.40.1
zathras-red     IN  A    10.1.10.1
zathras-blue    IN  A    10.1.20.1
zathras-green   IN  A    10.1.30.1
zathras-purple  IN  A    10.1.40.1
```

(MX info deleted for brevity, but I assure you they are awesome.)

Someone that wanted to reach any interface would use "zathras". A network engineer that needed to refer to the zathras interface on the "red" network would specify "zathras-red.example.com."

The reverse-lookup zone indicates the individual interface names (zathras-red, not zathras).

HostDB also has "ISMULTIHOMED=" which is the same thing but for servers. Currently it is exactly the same thing as "ISROUTER=". The separate directives are provided should different functionality be required for routers and hosts in future versions.

Host-hiding

Host hiding is where, for either real or perceived security reasons, most sites do not want to expose the

```
10.1.1.1    pot.example.com
10.1.1.2    kettle.example.com MAC=00:b0:d0:a6:cf:f1
10.1.1.3    spoon.example.com
10.1.1.4    fork.example.com   CNAME=pitchfork.example.com
10.1.1.5    spatula.example.com
```

Display 1: Syntax of hostdb.txt.

```
10.1.10.1   zathras-red.example.com       ISROUTER=zathras.example.com
10.1.20.1   zathras-blue.example.com      ISROUTER=zathras.example.com
10.1.30.1   zathras-green.example.com     ISROUTER=zathras.example.com
10.1.40.1   zathras-purple.example.com    ISROUTER=zathras.example.com
```

Display 2: Specifying .1 addresses as a router interface.

names of their hosts to the outside world. Inside their firewall, a host may be called patentdatabase.example.com, but outside it should be simply known as h64-32-179-55.example.com. Obscuring the host name like this prevents external users from being able to make educated guesses about good attack targets. It prevents internal host names from being exposed outside the network, which might prevent embarrassment by preventing ceo-pc.example.com from appearing in the Apache log files of www.sexyteens.com.

For forward DNS lookups one generally sets up an external DNS zone file that lists only the hosts that the public internet needs access to. This is used on external DNS servers and is often managed separately from the internal of DNS zones. Any time two separate databases are used to store information about the same thing you are asking for trouble; they will get out of sync. It is my experience that even sites with very full-featured commercial DNS management systems use a hand-edited zone-file for their external DNS zones.

Reverse DNS lookups are another matter. Some sites simply do not provide any reverse-DNS records. As a result, other sites refuse to talk to them because, for real or perceived security reasons, other sites are weary of accepting connections from sites without proper reverse DNS. Other sites simply use statically generated reverse labels that encode the IP address to keep them unique. For example, at this time Optimum Online's reverse DNS for 67.82.128.6 is ool-43528006.dyn.optonline.net (the digits are the hex interpretation of the IP address, the "ool" is Optimum OnLine).

To that end, HostDB generates a different set of zone files for "internal" DNS servers (those inside the firewall) and "external" servers (those accessible to the public). HostDB generates both sets of zone files from the same hostdb.txt, thus preventing them from getting out of sync.

To make this work, HostDB assumes that a host should be hidden from outsiders unless marked otherwise. To change the default, add a "scope qualifier" to the end of the hostname. The scope qualifiers are @EXTERNAL, @EXTERNALONLY and @INBOUNDNAT. Display 3 shows a hostdb.txt that demonstrates all three.

database.example.com is a normal host. Insiders have the usual forward and reverse DNS data. Outsiders can not look up the host's IP address, and the reverse DNS information is the anonymous name of d64-32-179-55.example.com. Lookups of d64-32-179-55.example.com return a proper A record.

www.example.com is an externally exposed host. Therefore, the forward and reverse lookups work as one would expect for both internal and external users. Since "example.com" is an ANAME for this host, it behaves the same way.

vpn3000 and vpn demonstrate the @EXTERNALONLY scope. This keyword exposes a host externally but not internally. It is rarely used. In the above example we use it for a VPN (RAS) server which external people use to access internal resources. Thus, they shouldn't use it when they are inside the company. We wanted to be able to construct a client configuration that would fail when used inside the firewall, where use of the VPN should be unneeded. Thus we marked vpn.example.com as @EXTERNALONLY so that the DNS entry would appear in external zone files, but not in internal zone files. This achieved the goal. However, network administrators still needed to access the VPN server for administrative reasons. They could use the name vpn3000.example.com which resolves properly internally.

The last two rows of the table demonstrate that reverse lookups for the anonymous hostnames are properly generated. Any A record that is generated needs a proper PTR record, even when host hiding.

Not pictured in Display 3 is @INBOUNDNAT which solves a very common, but specific, problem. Suppose you have an internal host on a RFC1918 address (unrouted) network and someone has decided to poke a hole in the firewall to let outsiders access it. The firewall will do some kind of "reverse NAT" so that packets destined to a particular public address will be re-written and sent to an internal host. While the author feels this has risky security implications, at

```
64.32.179.55     database.example.com
64.32.179.56     www.example.com@EXTERNAL ANAMES=example.com@EXTERNAL
64.32.179.57     vpn3000.example.com      ANAMES=vpn.example.com@EXTERNALONLY
```

Host	Internal DNS Records		External DNS Records	
	A	PTR	A	PTR
database.example.com	64.32.179.55	database.example.com	FAILS	d64-32-179-55.example.com
www.example.com	64.32.179.56	www.example.com	64.32.179.56	www.example.com
example.com	64.32.179.56	www.example.com	64.32.179.56	www.example.com
vpn3000.example.com	64.32.179.57	vpn3000.example.com	64.32.179.57	vpn.example.com
vpn.example.com	FAILS	vpn3000.example.com	64.32.179.57	vpn.example.com
D64-32-179-55.example.com	FAILS	FAILS	64.32.179.55	d64-32-179-55.example.com
D64-32-179-56.example.com	FAILS	FAILS	64.32.179.56	vpn.example.com

Display 3: Demonstration of scopes.

least HostDB lets you get the DNS records correct when you choose to use this technique. Here's how one would do this with HostDB:

```
64.32.9.8  host.example.com@INBOUNDNAT
10.1.1.5   host.example.com
```

Scopes can pertain to each hostname listed on a line, whether it is the host, a CNAME or an ANAME. A host can have multiple ANAMES and CNAMES and each name can be normal, external, externalonly, or inboundnat independently of the others. The generation process becomes quite complicated. What is the right PTR record to generate if a host is externalonly but has an ANAME alias that is normal? (Our answer: the ANAME on the inside, the externalonly name on the outside). As new situation cropped up we had to fine-tune the algorithms.

Mail Servers

Marking a host as a mail server means the MX records should be slightly different. It should have the default MX records but also an MX record pointing to itself that is better priority than the others. You specify this in hostdb.txt with the ISMAILSERVER flag; see Display 4.

HostDB does not create duplicates, even in tricky situations like when the default MX record for a subnet is the same as the host. For example, if zoneconf. txt specifies that all hosts are to receive MX records of priority 10 for mta1.example.com and priority 20 for mta2.example.com and hostdb.txt contains the code in Display 5, then the example.com zone file would resemble Display 6.

Host "normal" receives both MX records. "msexchange" receives the MX records and the best priority MX pointing to itself. "mta1" and "mta2" have the best priority pointing to itself with a worse priority MX pointing to either mta2 or mta1 as appropriate.

DHCP Pools

HostDB provides assistance to anyone creating large DHCP pools. DHCP pools usually assign "generic" names to each IP address such as ool-24-12-12-12.comcast.net. Repetitive jobs should always be automated, thus there is a syntax for automatically generating such hostnames. A DHCP pool in the middle of a subnet might look like Display 7.

This would generate a host name called dhcp-10-1-40-4-pool.example.com for the first DHCP_POOL entry, and similar entries for the rest.

```
198.65.112.13   mta1.example.com         ISMAILSERVER
```

Display 4: Mail server specification via "ISMAILSERVER".

```
198.65.112.9    normal.example.com
198.65.112.10   msexchange.example.com   ISMAILSERVER
198.65.112.11   mta1.example.com         ISMAILSERVER
198.65.112.12   mta2.example.com         ISMAILSERVER
```

Display 5: Mail servers with varying priorities in zoneconf.txt.

```
normal        IN A       198.65.112.9
              IN MX 10   mta1
              IN MX 20   mta2
msexchange    IN A       198.65.112.10
              IN MX 0    msexchange
              IN MX 10   mta1
              IN MX 20   mta2
mta1          IN A       198.65.112.11
              IN MX 0    mta1
              IN MX 20   mta2
mta2          IN A       198.65.112.12
              IN MX 0    mta2
              IN MX 10   mta1
```

Display 6: Resulting example.com zone file.

```
10.1.40.1    zathras-purple.example.com  ISROUTER=zathras.example.com
10.1.40.2    server2.example.com
10.1.40.3    server3.example.com
DHCP_POOL_TEMPLATE dhcp-$a-$b-$c-$d-pool.example.com
10.1.40.4    DHCP_POOL
10.1.40.5    DHCP_POOL
10.1.40.6    DHCP_POOL
10.1.40.7    DHCP_POOL
10.1.40.8    DHCP_POOL
10.1.40.9    DHCP_POOL
10.1.40.10   host10.example.com
```

Display 7: DHCP pool in the middle of a subnet.

The template takes effect from the point in the file it appears and continues until another DHCP_POOL_TEMPLATE line overrides it. For example, at one location they wanted to have slightly different hostnames for a DHCP pool for IP phones. In that address range, they put the lines shown in Display 8.

There is no coordination between the DHCP_POOL template and the DHCP configuration file that is generated. That is, if you use the DHCP_ POOL feature to generate a range of similarly-named systems, you must manually update the DHCP configuration file to include those ranges.

A future enhancement would be to have the template be a stack, permitting one to more easily switch to a new template and switch back when one is done.

While it might be useful to be able to specify a range of IP addresses (rather than to list one per line) this feature has been delayed until a clear syntax can be developed. A range would violate the "one line per address" rule that the sysadmins preferred. It turns out they often use their text editor's command to cursor-down 256 times to jump from subnet to subnet. Thus, any command that consolidates repetitive lines makes navigation more difficult.

However, a command-line tool is provided to generate lists of IP addresses with a template. The command was used to create the above range:

```
genrange 10.1.100.4 10.1.100.100 \
                          '$ip<tab>DHCP_POOL'
```

Zone Configuration

Zoneconf.txt stores everything required to generate the zones outside of the host information itself. Settings here change rarely and are usually the purview of senior DNS engineers. By keeping these options separate we avoid accidental modifications by junior engineers. The lines of this file are executed in order, like a sequential programming language.

The file usually starts by setting some options:

```
# When a hAAA-BBB-CCC-DDD external name
# is created, what domain is it in?
OBSCUREZONE example.com
```

OBSCUREZONE indicates the domain name to use for obscured, or anonymous, hostnames in external zone files.

SOA, MX, and ZONESERVERS sets the SOA values, default MX records, and NS records to be included in any zone file created after this point; see Display 9.

Notice that most commands are repeated once for INTERNAL and once for EXTERNAL. These settings are different for internal and external zones. There is also "SLAVE", which is an internal secondary. There is no support for external slave servers at this time as they tend to be at ISPs and don't let customers update their DNS server configurations directly.

When the system generates the named.conf file it will need to know some parameters to insert into any "zone" setting. For example, we need to specify who is allowed to do zone transfers and who is allowed to do dynamic DNS updates. We set those values as shown in Display 10.

Now the exciting part where we actually create some zone files. We do these with commands like DOMAIN and REVDOMAIN; see Display 11.

These lines direct HostDB to generate the zone file for corp.example.com, then example.com, then ex.com.

```
DHCP_POOL_TEMPLATE phone-$a-$b-$c-$d-pool.example.com
10.1.100.4  DHCP_POOL
...elided for space...
10.1.100.100     DHCP_POOL
DHCP_POOL_TEMPLATE dhcp-$a-$b-$c-$d-pool.example.com
```

Display 8: Example template for an IP phone pool.

```
SOA INTERNAL hostmaster.example.com 1H 15M 30D 60M
SOA EXTERNAL hostmaster.example.com 1H 15M 30D 60M

MX INTERNAL 10 lucy.example.com ; 20 ingw2.example.com
MX EXTERNAL 10 exgw2.example.com ; 10 exgw4.example.com

ZONESERVERS INTERNAL lucy3.example.com ingw2.example.com
ZONESERVERS EXTERNAL exgw4.example.com exgw2.example.com
```

Display 9: SOA, MX, and ZONESERVERS examples.

```
ALLOW-TRANSFER   INTERNAL        "primary_servers"
ALLOW-TRANSFER   SLAVE           "primary_servers"
ALLOW-TRANSFER   EXTERNAL        "rev65_servers"

ALLOW-UPDATE     INTERNAL        "none"
ALLOW-UPDATE     SLAVE           "none"
ALLOW-UPDATE     EXTERNAL        "none"
```

Display 10: Zone transfer and dynamic DNS update permissions.

Order is important here. After a host's DNS records are output they will not be output again. If example.com precede corp.example.com, the zone file for corp.example.com would be nearly empty, containing only the required SOA and NS records.

Notice that ex.com is only an external zonefile. That's permitted. Domains can be internal-only too. If you mix and match HostDB will do the right thing. This can be useful for subdomains that exist only inside the company.

Now we can generate the reverse lookup zone files; see Display 12. The REVDOMAIN options indicate what size in-addr.arpa zone file to create (class A, B, or C), the starting IP address, and whether this is internal only or both internal and external.

There are a few tricks one can do. For example, since the configuration is "executed" line by line, you can change settings between zones. For example, if you had different MX records for a particular domain, you can change along the way; see Display 13.

This would generate corp.example.com with lucy and ethel as the default MXs. Then the default MXs would changed to betty and wilma before prod.example.com is generated. Then the SOA defaults would be changed before the reverse domain 197.32.64.in-addr.arpa would be generated.

Rudimentary support for RFC 2317-style "classless" delegations is implemented but not complete. Eventually there will be a REVRANGE command that accepts a range like that shown in Display 14. This will generate a zone file with the specific contents required for RFC 2317.

Special Cases

While HostDB generates a file named.conf file it generates a very standard "zone {}" stanza for each DOMAIN command. If you need something very specialized, one can use CUSTOMDOMAIN instead. This will generate no "zone {}" stanza and rely on you to add one in manually using other means explained later.

This is particularly important for supporting Dynamic DNS updates. HostDB supports you by giving you enough rope to hang yourself.

Summary

That's all there is to it! You specify what zone files to create and it generates them plus the named.conf file. All the hard work is done for you. If extra lines need to be added to a zone file or configuration file, that can be done with the notation described in the next section. When the system is fully configured it should be able to generate all files without any manual intervention.

Header and Footer Files

HostDB can't be all things to all people, so there is an "emergency escape hatch" that lets you customize any file that is generated. You can add a header file and footer file to any file created by HostDB. For any file X generated by HostDB, the file that is actually created is made up of:

1. The contents of file $TEMPLATES/X-head (if this file exists).
2. The content generated by the HostDB system.
3. The contents of the file $TEMPLATES/X-tail (if this file exists).

```
DOMAIN corp.example.com      INTERNAL   EXTERNAL
DOMAIN example.com           INTERNAL   EXTERNAL
DOMAIN ex.com                           EXTERNAL
```

Display 11: Creating zone files using DOMAIN.

```
REVDOMAIN CLASSC 64.32.197.0   INTERNAL EXTERNAL
REVDOMAIN CLASSC 64.32.198.0   INTERNAL EXTERNAL
REVDOMAIN CLASSC 10.1.0.0      INTERNAL
REVDOMAIN CLASSC 10.1.255.0    INTERNAL
REVDOMAIN CLASSB 10.100.0.0    INTERNAL
REVDOMAIN CLASSC 10.254.0.0    INTERNAL
REVDOMAIN CLASSC 172.17.17.0   INTERNAL
```

Display 12: Creating reverse look up zone files using REVDOMAIN.

```
SOA EXTERNAL hostmaster.example.com 1H 1H 30D 60M
MX INTERNAL 10 lucy.example.com : 20 ethel.example.com
DOMAIN corp.example.com      INTERNAL   EXTERNAL

MX INTERNAL 10 betty.example.com : 20 wilma.example.com
DOMAIN prod.example.com      INTERNAL   EXTERNAL

SOA EXTERNAL hostmaster.example.com 2H 2H 30D 60M
REVDOMAIN CLASSC 64.32.197.0   INTERNAL EXTERNAL
```

Display 13: Specifying varied MX records for a particular domain.

```
REVRANGE 22.33.44.32 22.33.44.63 32-63.44.33.22.in-addr.arpa INTERNAL EXTERNAL
```

Display 14: Future style of RFC 2317-style range specification.

(The file name suffixes "head" and "tail" are used instead of "footer" and "header" because they sort better that way.)

This feature is useful for:

- Sneaking special records into a zonefile. If HostDB won't generate a zone the way you want it, add the lines to a -tail file.
- Adding custom zones to a named.conf file. If there are zones not under HostDB management you can still add information about them in named.conf
- Coarsely approximating an "include file" mechanism. If a DHCP server configuration file doesn't have an "include" mechanism, you can wrap the contents of a file around the output of HostDB's dhcpd.conf

DHCP Generation

HostDB parses the hostdb.txt file and creates the "host" statements appropriate for ISC's DHCP server. One stanza is created for each host that has a MAC= setting. No stanzas are created for hosts without MAC= settings.

A hostdb.txt line that looks like Display 15 would generate the default template like that shown in Display 16. This is appropriate for most systems. (Note that the MAC address is reformatted to be pretty by capitalizing the letters and zero-padding.)

However, if TYPE=foo is included, as alternate template is used. For example we might want a different template if we know the machine has a broken DHCP client that gets confused by extra data. A hostdb.txt line like this would select the template called "win95" which is intentionally minimal (see Display 17) generates the code shown in Display 18.

If the template includes a hyphen, the text after the hyphen is passed to the template as a parameter. Thus templates can be smart. Suppose there is a default netboot server and a special one for developers. Templates can be constructed that produce proper DHCP configuration for each; see Display 19.

Currently the templates are written as subroutines in Perl and are hard coded into the mkdhcp script. Future versions will move these templates to their own files, though this work has been deferred to later because (1) the need for special templates to handle broken DHCP clients is reduced over time as more vendors figure out how to write DHCP clients that aren't broken, (2) nobody's really complained about the system as it is.

Deployment (Phear My Aw4z0m3 Makefile Skilz!)

Generating zone files and configuration files is not enough. They must be deployed to be useful.

Rather than generate files directly where they will be used, HostDB generates all files in a directory called "out" (short for output). Sites with a very large legacy system for DNS/DHCP management can simply copy the newly generated files (or just the files that are useful to them) out of this directory. However, HostDB includes a tool that makes this even easier called mkdestination. It generates a Makefile that "does the right thing."

Mkdestination takes "destination.txt" for input, and generates "destination.mk" which can be used in the Makefile that drives your DNS/DHCP service.

Consider short example destination.txt file shown in Display 20. The format is a simple list source files, the "->", then destinations. Destinations

```
10.10.244.14 dell4.example.com MAC=00:5:3b:F1:21:7a
```

Display 15: Sample hostdb.txt file.

```
host dell4.example.com {
       hardware ethernet 00:05:3B:F1:21:7A;
       fixed-address 10.10.244.14;
       option host-name "dell4.example.com";
       ddns-hostname "DELL4";
}
```

Display 16: Generated default template from hostdb.txt file above.

```
10.10.244.14 dell4.example.com MAC=00:5:3b:F1:21:7a TYPE=win95
```

Display 17: Minimal template for "win95".

```
host dell4.example.com {
       hardware ethernet 00:05:3B:F1:21:7A;
       fixed-address 10.10.244.14;
}
```

Display 18: Code generated by template above.

```
10.10.244.13   human1.example.com MAC=00:05:3B:F1:21:7A TYPE=netboot
10.10.244.14   human2.example.com MAC=00:05:3C:E3:32:A4 TYPE=netboot-main
10.10.244.15   human3.example.com MAC=00:05:32:A2:01:12 TYPE=netboot-dev
```

Display 19: DHCP configuration templates.

can be a file name or directory on the local machine or some other. There is no macro expansion or other substitution mechanism because we want to keep the format simple.

The first line says that to deploy INTERNAL.example.com and INTERNAL.198.32.64.in-addr.arpa one should copy it into the local directory /var/named, and scp it to a /var/named on a machine called "betty". The next two lines specify that a file is to be copied to a particular filename. This permits one to rename files as they are copied.

Display 21 shows how one might use mkdestinations in a master Makefile.

Invoking "make push" would generate the destinations.mk file (if needed) then run the "all" recipe. The recipes in the destinations.mk file "do the right thing" for all files mentioned in destinations.txt. "make push-local" is similar but runs the "all-local" recipe, which only deploys files destined for the local machine. This is useful for testing purposes.

Mkdestinations is needed to overcome limitations in "make". "Make" has no awareness of remote file timestamps, nor can "make" handle a dynamic list of zones and files with multiple destinations.

Efficient Updates of Remote Files

Make has no idea if betty:/var/named/named.conf is up to date because it is on a different machine. Therefore, mkdestination compensates by creating local timestamp files after a remote operation is complete (that is, after a remote file is updated successfully it touches a local file whose name encapsulates the destination server and filename). Future timestamp comparisons can be done against the local file. Rather than recopying a file to a remote machine "just in case", the timestamps can be compared. In the case of copying a file to betty:/var/named/named.conf, the timestamp file ds/ betty__var_named_named.conf is created. (The actual filename is much more complicated to encode special characters that might cause problems.)

Mkdestination also creates a recipe called "clean" which deletes all timestamps. Thus, to force a "push" to all other hosts, simply "make clean" then "make push" and all files will be copied whether they need it or not.

Efficient Zone Serial Number Updates

The other interesting feature of mkdestination is that it is smart about zone serial numbers.

DNS zone files have an embedded serial number which is used to determine when zone transfers are to happen. If a secondary has a zone of serial number X, and is told that the primary has a higher serial number, it requests a download of that zone (a "zone transfer"). Zone transfers are a CPU- and (possibly) network-intensive operation. We want to prevent gratuitous zone transfers.

The problem, however, is that the only way to know if a zone file will change is to generate the new one and compare it to the old one. The comparison can't be a simple "cmp" or "diff" because the serial number inserted into the zone file will be different every time. Most changes to hostdb.txt tend to affect only a few zone files.

Again, mkdestination to the rescue. mkdestination knows about serial numbers and can do comparisons that ignore the serial number line of a zonefile. The methodology is as follows.

HostDB generates all files in a directory called "out". Any file that needs a serial number contains the text ":serial:" anywhere a serial number is to be placed. This makes comparison of zone files easier. Any file that did change is copied to a directory called MP and another copy is placed in a directory called MS. The MP directory contains a plain copy of the zone but the MS directory contains a copy of the file with ":serial:" changed to the serial number used for all files generated at that time. When deciding which files need to be deployed, the timestamp of MS is used to determine if the file needs to be copied but the file that is copied comes from MP.

After a file is successfully deployed, a timestamp file is created in the DS directory marking the comple tion. In the case of copying a file to betty:/var/named named.conf, the timestamp file with a name like DS/ betty__var_named_named.conf is created. The result is

```
INTERNAL.example.com INTERNAL.198.32.64.in.addr.arpa -> \
                                  /var/named/. secondary:/var/named/.

INTERNAL.named.conf -> /etc/named/named.conf

SLAVE.named.conf -> betty:/etc/named/named.conf
```

Display 20: Sample destination.txt file.

```
destinations.mk: ../destinations.txt
      mkdestinations <../destinations.txt >destinations.mk

push: destinations.mk
      make -f destinations.mk all

push-local: destinations.mk
      make -f destinations.mk all-local
```

Display 21: Using mkdestinations in a Makefile.

that files are only copied when absolutely needed. It is optimal.

At a site with dozens of zones the DNS servers were being hit fairly hard by all the zone transfers. After mkdestination's technique was deployed the administrators were very happy to see that very few zones were actually being transferred after typical updates. It was quite impressive to see many, many zone files with an assortment of serial numbers as diverse as the history of updates had dictated. Yet, no tool more complicated than "mkdestination" and "make" was needed.

If this sounds confusing to you, just be happy in the knowledge that you don't have to understand it to benefit from the efficiencies. Files that don't need to be copied across your network aren't copied.

Deployment Examples

Here are two examples of how to deploy HostDB. First we will consider a new site starting from scratch. Then we will see how easy it is to deploy in a pre-existing environment.

We need a place to put our files. The HostDB files can be put in any directory as your site's requirements dictates (/var/hostdb or /home/adm/hostdb is typical). HostDB assumes that all executables will be found in the shell's PATH. As a result, test datasets can be placed anywhere (even /tmp). When testing new executables, one can simply put the newer versions ahead of the older ones in the shell's PATH.

In these examples we will use /var/hostdb and assume the PATH is set correctly.

Example 1: Starting from Scratch

In this example we are starting from scratch. Imagine we are setting up a new network for a new company. We have one external network (64.32.179.0/24) and one internal network (10.1.1.0/24). There will be a dhcp pool from .140 to .250 in the first network.

We begin by generating the hostdb.txt file. We generate a list of each IP address. That way when a junior engineer goes to allocate an IP address, they don't have to understand anything but (1) find an unused address on the right subnet, (2) remove the comment symbol and change the hostname as appropriate. This prevents them from introducing typos into the list of IP addresses, and prevents mistakes such as thinking they can create new IP subnets by simply editing this file. It also keeps the file neat and orderly in case you work with people that can not be trusted to keep a file in numerical order.

Display 22 shows commands that will generate a hostdb.txt that is a good starting point.

- Line 1 creates commented-out sample lines for the first 140 addresses.
- Line 2 creates addresses for our DHCP pool. Note the custom template.
- Line 3 completes the internal network's addresses.
- Line 4 uses a different template for the external hosts. It also demonstrates that if the second IP address is replaced by an integer, it is interpreted as a count of how many lines to output instead of an end address.

Display 23 shows the output.

Now edit the hostdb.txt to include the initial hosts that will be installed. To make sure that nobody

```
genrange  >hostdb.txt -d example.com 10.1.1.0   10.1.1.139
genrange >>hostdb.txt -d example.com 10.1.1.140 10.1.1.250 '$ip\tDHCP_POOL'
genrange >>hostdb.txt -d example.com 10.1.1.251 10.1.1.255
genrange >>hostdb.txt -d example.com 64.32.179.0 256 '#$ip\tex$hex.$DOMAIN\@EXTERNAL'
```

Display 22: Generating hostdb.txt from scratch.

```
#10.1.1.0        UNUSED0A010100.example.com
#10.1.1.1        UNUSED0A010101.example.com
...elided for space...
#10.1.1.138      UNUSED0A01018A.example.com
#10.1.1.139      UNUSED0A01018B.example.com
10.1.1.140       DHCP_POOL
10.1.1.141       DHCP_POOL
...elided for space...
10.1.1.249       DHCP_POOL
10.1.1.250       DHCP_POOL
#10.1.1.251      UNUSED0A0101FB.example.com
#10.1.1.252      UNUSED0A0101FC.example.com
#10.1.1.253      UNUSED0A0101FD.example.com
#10.1.1.254      UNUSED0A0101FE.example.com
#10.1.1.255      UNUSED0A0101FF.example.com
64.32.179.0      ex4020B300.example.com
64.32.179.1      ex4020B301.example.com
...elided for space...
64.32.179.254    ex4020B3FE.example.com
64.32.179.255    ex4020B3FF.example.com
```

Display 23: Output from above commands.

tries to allocate a broadcast address, we mark the "all zeros" address as "foo-net.example.com" and the "all ones" address as "foo-bcast.example.com" where "foo" is a unique name for each subnet. Display 24 shows the non-comment lines.

Display 25 shows our zoneconf.txt file, which is almost the same as the example provided with the software. Most sites can use this template and simple change "example.com" to their domain and edit their MX records and ZONESERVERS to suit their needs.

The last three lines show which zones to generate. First the zone for example.com, which has different values for the internal and external version. Next we generate the reverse lookup zone for 10.1.1.0/24, which only generates an internal zone file. Lastly we generate the 64.32.179.0/24 zone reverse lookup which has different internal and external values.

Copy the template Makefile into the main directory and we are ready for our first attempt at generating our zones. To save ourselves typing, we'll create an alias for bash/sh/ksh or csh/tcsh (see Display 26). Then we can do "makeh" to generate a new set of files. If we are happy we can do "makeh push" to deploy the files. As a result, the "out" directory now contains:

```
10.1.1.0        main-net.example.com
10.1.1.1        zathras-main.example.com ISROUTER=zathras.example.com
10.1.1.2        fileserver.example.com
10.1.1.3        mailserver.example.com   ISMAILSERVER
10.1.1.4        vector.example.com
10.1.1.10       staffpc1.example.com MAC=00:b0:d0:a6:cf:f1
10.1.1.11       staffpc2.example.com MAC=00:b0:d1:a7:c0:d1
10.1.1.11       staffpc2.example.com MAC=00:b0:c2:b3:c3:d3 TYPE=freebsd
10.1.1.140      DHCP_POOL
10.1.1.141      DHCP_POOL
10.1.1.142      DHCP_POOL
...elided for space...
10.1.1.249      DHCP_POOL
10.1.1.250      DHCP_POOL
10.1.1.255      main-bcast.example.com@EXTERNAL
64.32.179.0     ext-net.example.com@EXTERNAL
64.32.179.1     isp-router.example.com@EXTERNAL
64.32.179.2     zathras-ext.example.com@EXTERNAL ISROUTER=zathras.example.com@EXTERNAL
64.32.179.3     mailqueue.example.com@EXTERNAL ISMAILSERVER
64.32.179.4     vector.example.com@INBOUNDNAT
64.32.179.5     exweb.example.com ANAME=www.example.com@EXTERNAL
64.32.179.255   ext-bcast.example.com@EXTERNAL
```

Display 24: Edited hostdb.txt file.

```
TEMPLATEDIR ..
OBSCUREZONE example.com

SOA INTERNAL hostmaster.example.com 3h 1h 1w 1h
SOA EXTERNAL hostmaster.example.com 3h 1h 1w 1h

MX EXTERNAL 10 mailserver.example.com
MX INTERNAL 10 mailqueue.example.com : 20 s1.isp.com

ZONESERVERS INTERNAL fileserver.example.com mailserver.example.com
ZONESERVERS EXTERNAL mailqueue.example.com exweb.example.com

ALLOW-UPDATE   INTERNAL        done
ALLOW-UPDATE   INTERNAL        none
ALLOW-UPDATE   SLAVES          none
ALLOW-UPDATE   EXTERNAL        none

DOMAIN example.com              INTERNAL EXTERNAL
REVDOMAIN CLASSC 10.1.1.0       INTERNAL
REVDOMAIN CLASSC 64.32.179.0 INTERNAL EXTERNAL
```

Display 25: Example zone.txt file.

```
# bash/sh/ksh alias:
alias makeh='cd /var/hostdb/out && make -f ../Makefile'

# csh/tcsh alias
alias makeh 'cd /var/hostdb/out && make -f ../Makefile'
```

Display 26: Shell 'makeh' alias definitions.

```
$ ls -1
EXTERNAL.179.32.64.in-addr.arpa
EXTERNAL.example.com
EXTERNAL.named.conf
EXTERNAL.named.root
INTERNAL.1.1.10.in-addr.arpa
INTERNAL.179.32.64.in-addr.arpa
INTERNAL.example.com
INTERNAL.named.conf
SLAVE.named.conf
```

Notice that each filename is prefixed with "INTERNAL.", "EXTERNAL." or "SLAVE.". All other files are tagged as being appropriate for either the inside or the outside. EXTERNAL.named.root is the NIC's "root cache" file which the Makefile automatically retrieves via FTP.

Now check the zone files by manual inspection. Correct any errors by editing ../hostdb.txt and re-running "makeh" until you are satisfied.

Notice that in addition to zone files, ISC BIND named.conf files are generated. If you examine them, you'll find them very anemic. They just contain the "zone {}" entries which are automatically generated. To create a complete file, determine what you would put before and after the generated parts and put them in the proper -head and -tail files. Any text in ../INTERNAL.named.conf-head is prepended to the generated file, and any text in ../INTERNAL.named.conf-tail is appended to the end of the generated file. The same is true for all files generated by the system.

Now that we are happy with the files being generated, it's time to tell HostDB where to put them. To do that, we create ../destinations.txt which lists where each file is to be copied; see Display 27.

- Line 1 specifies that EXTERNAL.named.conf is to be copied to mailqueue.example.com in a directly called /etc/namedb. It is renamed to named.conf as it is copied. It is also copied to exweb.
- Line 2 specifies three files that are to be copied to two servers in their /var/named directories. The files are not renamed as they are copied. Instead, we will make sure the configuration files that refer to these files must specify the filename as they exist.
- Line 3 is like line 1 but for the internal servers.
- Line 4 is analogous to line 2.

This command will generate a makefile that will do the actual copying: makeh destinations.mk. Finally, "makeh push" will actually copy the files into place. A few rounds of debugging and the system is deployed.

Example 2: Replacing a Legacy System

In this scenario we have a working system where we hand-edit zone files and copy them using a shell script. To move from a legacy system to HostDB requires a lot of testing. HostDB includes utilities that help every step.

The first and most difficult step is to convert the zone files and turn them into hostdb.txt format. This is the reverse of what HostDB does! Sadly, we can't reverse the polarity of the flux capacitor and have everything "just work." Luckily, the HostDB package includes zone2hostdb which does 90% of the work for you. It takes a zone file as input:

```
zone2hostdb <zonefile >hostdb.txt-base
```

Since this step is automated we reduce the potential for mistakes considerably.

This gives you a first draft of what hostdb.txt should be. Now check these issues:

- Verify all mail servers are listed as ISMAILSERVER
- Verify that all routers and multihomed hosts are marked as ISROUTER or ISMULTIHOMED as appropriate
- Mark any broadcast addresses so they are not accidentally used
- Check any ANAMEs to verify that the official name is listed first on the line and that actual aliases are listed in an ANAME. This assures proper PTR records
- If any special cases are in place for PTR records, make sure the name to be used for the PTR record is always the first hostname on the line
- Set the scope to @EXTERNAL for any hosts accessible externally
- Set the scope to @INBOUNDNAT for any host that is accessed by external users by a different address than internal users
- Set the scope to @EXTERNALONLY for any hostnames only external users should access (very rare)
- Anything else "special" about your DNS zones

```
EXTERNAL.named.conf -> mailqueue:/etc/namedb/named.conf  \
                       exweb:/etc/namedb/named.conf

EXTERNAL.example.com EXTERNAL.179.32.64.in-addr.arpa \
                EXTERNAL.named.root -> \
                mailqueue:/var/named/. exweb:/var/named/.

INTERNAL.named.conf -> fileserver:/etc/namedb/named.conf \
                       mailserver:/etc/namedb/named.conf

INTERNAL.example.com INTERNAL.1.1.10.in-addr.arpa \
   INTERNAL.179.32.64.in-addr.arpa -> fileserver:/var/named/. \
   mailserver:/var/named/.
```

Display 27: File ../destinations.txt: where files are to be copied.

The next step is to use genrange to enumerate every IP address that should be listed in your hostdb.txt file, whether it is commented out or not. We use the same "genrange" commands as in the first example except we save the output to a file called hostdb.txt-enum.

HostDB includes a utility called "mergeiplists" that will merge these two lists properly even though some of the lines are commented out. It assumes the first appearance of an IP address is the authoritative line and throws the others away. Thus, we list the hostdb.txt-base first because our hand-edited file should be authoritative; hostdb.txt-enum is just to fill in the gaps; see Display 29. Now we have a pretty decent draft of our hostdb.txt file.

Before we can put it into use, we should test it extensively and then only switch to it incrementally.

Testing is made easier by a utility that is included called "canonzone". Comparing two zone files can be difficult because of small changes in white space, sorting, and so on. "canonzone" reads a zone file and outputs it in a very specific, clean, regular, format (a canonical format). If you are comparing a legacy zone file and a generated zone file, pass both thought canonzone first and the comparison will be much easier. The distribution includes a file called "Example_comparezones" which takes two files, passes them through canonzone, strips them down to just DNS A records, and compares them.

Here are some tips for testing your new zones:

- Use Example_comparezones as a template for your own comparison scripts.
- Review each email server and verify the MX records are as required.
- Review all other servers and verify their DNS records are as expected.
- Review each external host and verify all of its records (test from inside and outside).
- Set up a new DNS server and load the zones. Make sure they load without outputting any errors or warnings in BIND's logfile.
- Query the DNS server to make sure you get expected results for mail servers and other important hosts. When you are satisfied, ask co-workers to do the same.
- When you are satisfied, configure yourself and a few trusted users' machines to use this new DNS server. If anything breaks, fix it. After a few days, have your coworkers switch to this DNS server. Make sure they know to report problems to you.

When you are satisfied with the zone files, you can slowly migrate them into your legacy system. The destinations.txt file can be used to roll out just the specific zones you are confident in.

Suppose your legacy system loads your main zone from a file called /var/named/zone.example.com..zone, you can replace that file with the HostDB-generated file with a statement like that shown in Display 28. Obviously you should make good backups before deploying any new zones.

To push that zone out, "makeh push".

Test, test, test. Be ready to revert to the legacy zone if the problems you find are insurmountable. Follow any in-house procedures related to change windows and so on. For example, do not introduce new zones:

- Right before you leave for vacation
- The same week as the quarterly results or taxes are being prepared
- The week of a major deadline for your company
- The week raises and promotions are being decided

Lastly, you can replace your named.conf files with the ones that are generated by HostDB (though if your zone filenames change, be very careful!). You should also integrate the Makefile into your system so that it controls all DNS-related activity from one central point.

Related Work

There are many other open source systems that do some or part of what HostDB does. There are even a few that do a lot more, especially where dynamic registration is involved. However, the goal of HostDB is to maximize features without having to resort to requiring a web front-end or SQL backend. The author believes HostDB is superior to all systems that don't require an SQL backend, and easier and faster to install than any of the systems that require SQL databases.

Here is my comparison to some popular utilities:

- h2n – O'Reilly's "DNS and BIND" by Paul Albitz and Cricket Liu comes with h2n (and an enhanced version is called h2n-hp). These are fine DNS generator scripts. While it can be used for one-time conversion, there are features that let you maintain the host list as a /etc/hosts-like file. However, this is a "kit", not a complete system. It does not include anything like mkdestinations for deployment. No DHCP support. Source: www.oreilly.com/catalog/dns3 .
- DNSDusty – While this system does automate deployment of the files it generates, it requires a web interface (but no SQL database). It makes adding a host very easy for a junior sysadmin once it is set up. No DHCP support. Source: www.poochiereds.net/dnsdusty .

```
INTERNAL.example.com -> dnsserver:/var/named/zone.example.com..zone
```

Display 28: Replacing a legacy zone file.

```
mergeiplists hostdb.txt-base hostdb.txt-enum >hostdb.txt
```

Display 29: Merging lists.

- dnscvsutil – Maintains your DNS zone files under CVS control and automatically updates reverse zones. Helping someone do a better job of maintaining hand-edited zone files is exactly the opposite of what I want to encourage. Though, revision control is better than no revision control. No DHCP support. Source: freshmeat.net/projects/dnscvsutil .
- Sauron – Sauron is a free DNS/DHCP management system with Web and command line interfaces. It can manage multiple servers and generates complete dhcpd and named configurations from the database. This is a very complete and impressive package. It manages deployment of generated files. However, it does require a web interface and SQL database. Source: http://sauron.jyu.fi/ .
- Updatehosts – A very comprehensive system for managing DNS information. This is popular at many large commercial sites. However, the learning curve seems overly steep as it requires understanding of database relations. This is not for the neophyte or someone trying to set up a new site quickly. Requires SQL. No DHCP support. Source: ftp://ftp.tic.com/pub/updatehosts .

The Future

New features are added to HostDB on a regular basis.These projects are being considered:

- Make testing even easier: mkdestination should create a "diff" recipe that diffs the last distributed against the most recently generated files.
- Eliminate the need to modify the default Makefile: Move "restart commands" to the mkdestination system.
- Add classless delegation support: Complete support for RFC 2317-style "classless" delegations.
- Add IPv6 support: Add support for IPv6 records. (Those long addresses aren't going to be fun to type. We need some human-factors help here.)
- Improve legacy conversion: Update zone2hostdb to read multiple zones, including reverse DNS zones, and use the information to do a better job. In particular, PTR records could be an indicator of which name is a canonical name and which is an ANAME.

Conclusion

HostDB generates really cool DNS zone files with all the features required by sites with complicated DNS configurations, such as aliases, "external only" names, "host hiding" and multihomed hosts. It also generates the ISC BIND configuration files required for primaries, secondaries, and external DNS servers. DHCP configuration for static leases are generated for ISC DHCP using a very flexible template-based mechanism. It is easy to configure and deploy, yet sophisticated behind the scenes. It does not offer complicated features such as real-time host authorization, web front-ends, or any feature that required sophisticated and difficult to maintain databases.

Deploying the files is made easier through the use of a simple Makefile that is generated for you. The updates are very sophisticated, doing the minimal number of updates locally and to remote machines, even being careful not to create "zone transfer storms" by not updating a new serial number for a zone that does not warrant it.

Availability

The software open source and is available at http://www.everythingsysadmin.com/hostdb on the web.

Acknowledgements

I would like to thank Glenn Sieb for his feedback when designing the file formats, David H. Potter for assistance developing the mkdestination algorithm, Joe Gross who read the most painful early drafts of this paper and helped me turn it into a much better paper, and Josh Simon for his editing and proofreading assistance.

References

Albitz, Paul, Cricket Liu, *DNS and BIND, Fourth Edition*, O'Reilly, April, 2001.

Cheswick, William R., Steven M. Bellovin, Aviel D. Rubin, *Firewalls and Internet Security: Repelling the Wiley Hacker, 2nd Edition*, Addison-Wesley, February, 2003.

Christiansen, Tom, Nathan Torkington, *Perl Cookbook, Second Edition*, O'Reilly, August, 2003.

Limoncelli, Thomas A. and Christine Hogan, *The Practice of System and Network Administration*, Addison-Wesley, 2002.

Wall, Larry, Tom Christiansen, Jon Orwant, *Programming Perl (third edition)*, O'Reilly, July, 2000.

Solaris Service Management Facility: Modern System Startup and Administration

Jonathan Adams, David Bustos, Stephen Hahn, David Powell, and Liane Praza
– Sun Microsystems, Inc.

ABSTRACT

Application uptime is critical to every administrator. The factors which cause system downtime are often handled by the operating system, but causes for application faults (e.g., software bugs, hardware faults, or human errors) are not addressed by standard system software. Recovery is left to humans, who may often compound the problem due to misdiagnosis or simple error. While availability issues have traditionally been addressed by expensive high-availability clustering solutions, the increasing complexity of software stacks requires a solution for all systems.

In addition to the challenges of managing availability of higher level software, the modern operating system itself is composed of many interdependent software entities. A failure in any one of these components often cascades, causing failures in other components. A complex software model with many interdependent elements makes diagnosing failures very challenging for system administrators. The traditional init.d script mechanisms for UNIX are only a weak reflection of the intricate dependency relationships which exist on every system.

We introduce the Service Management Facility (SMF) as a comprehensive way to describe, execute, and manage software services. SMF promotes the service to a first-class operating system entity, without requiring modification of application binaries or changes to the UNIX process model. It relieves the administrator from duties of application failure detection and restart, and provides sophisticated diagnosis tools when automatic repair is impossible.

Introduction

As software running on a system becomes more complex, the traditional separation of management of system startup from run-time management becomes untenable. The reliability of modern hardware, coupled with the ever-increasing complexity of modern software means that applications are just as likely, if not more likely, to be the point of failure on a system. Reboots are costly, especially on sophisticated hardware.

A system administrator is expected to ensure that critical applications are always running. This is sometimes done by manual monitoring or an expensive to maintain home-grown tool, or the administrator is expected to turn to software not well integrated with the operating system – either sophisticated monitoring packages, or expensive and often complex high-availability clustering software. The unbundled solutions usually also focus on higher level applications, and can neglect potential failures of core operating system daemons.

But, even higher level applications do not stand on their own. In order to provide business-critical functionality, often two or more pieces of software must be working and cooperating closely together. Failure of even a less critical application can have unforeseen consequences in important software.

In addition to basic availability concerns for software services, the service management model needs to be significantly enhanced. Without a common model and interface, it is very difficult to ask the system even basic questions such as: "what's running?", "what's broken?", and "what applications are available on this system?".

Application management is a fundamental task of all administrators: managing applications should not require a bolt-on solution. Driving the service management interface into the operating system also brings significant benefits through encouraging evolution of core operating system functionality to support the service, and creating a truly common interface for all user software: operating system daemons and third party products alike.

The UNIX service management model has not evolved significantly beyond the traditional process model coupled loosely with service startup scripts, and is stretched in trying to meet modern system administration demands. The lack of functionality is costly for administrators, who must spend significant effort monitoring and managing systems that provide no fundamental abstractions to allow this.

SMF directly addresses all of these administration gaps and provides an integrated solution for service delivery and management on UNIX systems. SMF defines a fundamental service model which is used to control system startup, introduces a management interface for system and application services, and delivers diagnosis and restart capabilities for all services to maximize application availability.

Background and Related Work

The traditional System V and BSD style init.d/rc systems provide an extremely flexible mechanism for initiating arbitrary processes during system startup. For all of the flexibility provided, a number of deficiencies are apparent. We'll enumerate these deficiencies in terms of the System V implementation, but analogous problems exist in the BSD style system.

- No systematic way to list services. As the init.d system does not even encourage, much less enforce a 1:1 mapping of system services to init.d scripts, a simple ls of rc directories is never sufficient to get a full list of services provided by the system. The services required to start core operating system components are often the most opaque to the administrator.
- No persistent mechanism to indicate to the system whether a service should be running or not. Upgrade of software components often updates or even re-creates the corresponding rc script, overwriting administrative customizations such as attempts to disable the service.
- No formal dependency management interfaces. System components are usually very sensitive to ordering changes. The System V lexicographical ordering provides some latitude for the third-party service. Service developers can have their script run at a certain point during startup (before some services and after others) by giving the script an appropriate name. But, the lack of explicit dependency specification for a service makes modifications to the ordering of core system services fragile, and leads to unexpected behaviour.
- Incomplete manual restart capabilities. The individual init.d scripts are not necessarily stand-alone or idempotent. The administrator is not expected to execute all of the init.d scripts manually, so there is no simple way to restart many individual services without understanding the intricacies of their implementation. Closely cooperating services compound the problem; restart of one may require restart of its closely coupled dependent services. The init.d system provides no mechanism to determine service interdependencies, so significant administrative expertise is required to restart individual services without a reboot.
- Nonexistent automatic restart capabilities. The init.d system does not include automatic detection and restart of failed components. Detection and restart is left to orthogonal mechanisms such as inittab and inetd.conf.
- Lack of correlation between processes and services. Services are becoming increasingly sophisticated; the traditional UNIX model of a single daemon process as the entirety of a service is no longer adequate. Multiple processes are correlated into service boundaries by administrative expertise rather than being programmatically represented as a first-class object. The UNIX process model complicates the analysis – daemons are often reparented to init and lose their parent-child association in the process tree.
- Ticking configuration time bombs. There is no formal association between the current status of the system and the expected state when the system is restarted because starting a service has no impact on its settings on reboot. In addition, there is no systematic way to determine current versus configured state, so a reboot often leads to unpleasant surprises due to un-configured or mis-configured services.

For rigidly specified service models, such as inetd, the deficiencies can be more easily addressed. However, a complete system management architecture must provide a model sufficiently rich for all system services. SMF addresses each of these deficiencies for general software services.

Previous work in this area aims primarily to improve deficiencies of the init.d/rc system in system startup and manual service enable, disable and restart, up to and including a more formal statement of dependencies [1, 2, 3]. This work helps administrators gain better visibility into system service configuration and provide slight enhancements to administrative tools. However, if application availability is the crucial measurement, these sorts of enhancements only aid insofar as they slightly reduce the chances of administrative error.

Some recent efforts [4, 5] are more sophisticated and include simple service restart. The service can be restarted either when requests for that service come in, or restarted when the service exits unexpectedly. However, these efforts assume a single process is the primary provider of service, which is insufficient for a large class of business-critical applications.

Design Principles

Altering system service management is guaranteed to cause a change in administrative procedures. Changing something as fundamental as the init.d system requires administrators to spend time learning the new management paradigm. Thus, we took an ambitious approach so that the significant changes happened all together, rather than creating an incremental education cost over many software releases. The redesign of service management procedures must be comprehensive, and consider the full range of administrative procedures, software errors, and hardware errors.

The most fundamental requirement of this work is a simple, common model for all services. Services must be able to encompass nearly all capabilities of the system – simple daemons, network services, complex databases, and even non-process-based services such as dump device configuration. The management framework must

be flexible enough to allow common actions to be done easily and with no required administrative knowledge of the service implementation. A common model for all application and system services is required in order to thoroughly leverage administrator expertise; an administrator should be able to manage syslogd using the same basic commands as telnetd.

The service model must be extensible. New types of services, increased diagnosability, and expanded meta-configuration should be easy to support with no required enhancements to existing service descriptions.

Relationships between services are more complex than start time requirements. Many services are written carefully to be resilient to failure of their dependencies. For example, a well written networking application does not need to be restarted when a transient nameservice failure occurs. However, starting even a defensively-coded application without nameservice availability is usually futile. Thus, a rich dependency declaration is required to encompass both startup and failure scenarios.

Services then are restarted according to their dependencies following either hardware or software failure. A service restart must be attempted following any type of failure, such as a critical process exit, an uncorrectable memory error or other hardware error, a software coredump or premature exit, or an administrator accidentally killing the wrong process. Any dependents of that service must be restarted if they specify intolerance to dependency failures.

When a service cannot be automatically repaired after a fault, the system must provide significant aids to manual diagnosis, and trace failure back to the responsible component. When software failures can be traced back to the initial fault, this information as well as as much data as possible about the specific fault must be provided to the administrator so that he can spend his time repairing the fault rather than debugging an error in a dependent service. For example, if an application cannot start because the filesystem containing the application data was unable to mount, the system should point to the filesystem as the root cause of the fault, rather than incorrectly indicting the application.

The other critical component of a service management interface is its handling of service configuration data. We identified a set of meta-configuration, which is similar across a large set of services. For example, one critical pieces of meta-configuration captures whether a service is supposed to be running on a specific system. Common tools should be provided for meta-configuration, while still retaining the flexibility for services to provide their own complex application-specific configuration.

Meta-configuration must be available via a common command set and API across all services. This allows rapid development of higher-level administration tools which need no application knowledge to perform common tasks. Service-specific configuration can also be stored in a format accessible by this API, but the choice to transition to the new configuration store is up to the application developer depending on their compatibility requirements. Through the common commands, configuration rollback is provided. An administrator should be able to undo configuration or meta-configuration changes which rendered a service unstartable.

A few constraints were also required for a practical implementation in an operating system with strong compatibility requirements:

- No application binary changes required for basic participation in the framework.
- Compatibility for existing init.d services. The vast majority of services must continue to work, to account for lag time in new feature adoption by software developers.

Extending UNIX Processes to Software Services

SMF formalizes software "services", and introduces the tools to create, observe, manage and automatically restart them. A service is a long-lived software object (typically, a daemon) with a name, a clear error boundary, start and stop methods, and dependency relationships to other services on the system.

A service may be composed of zero, one, or many processes. A comprehensive software service model must allow all of these process models, and take each into account when defining failures. We'll first consider a typical UNIX service which only has one process. These services are relatively straightforward to manage and monitor. The single process makes process-to-service mapping easy to understand, as no complex lineage from parents to children needs to be tracked. If the process exits, the service is no longer being provided and must be restarted. Still, without a new mechanism to track process faults, most monitoring implementations would require services forgo the traditional reparent-to-init step.

Even simple services can consist of multiple processes. For example, the Sendmail service usually includes two processes. Failure of either process should cause the service to be restarted. Traditional models for determining the relationship between these two processes fall down: they both re-parent themselves to init, breaking monitoring implementations which rely on the parent-child relationship. Aside from the process name, it is difficult to tell they're both part of the same service.

Finally, we have a set of services which only appear transiently during the startup process. Usually, they exist to execute a command or small set of commands which change configuration state, such as informing the kernel about certain configuration parameters. It is important for SMF to understand a priori that a lack of long-running processes in these *transient* services does not constitute a fault.

In order to monitor the health for all three types of services, SMF introduces a kernel interface called a contract. A *process contract* allows a userland process to register an interest in a process and all of its children. The process receives reliable events when important changes occur, such as when all processes in the contract exit, a hardware error occurs in any of the processes, any processes in the contract coredump, or any process in the contract receives a fatal signal. All processes on the system must belong to a process contract, and all children of a process are part of the same contract until one creates a new contract.

Service *restarters* are responsible for managing software services. A restarter can use process contracts to receive and respond to fault events for a service. Usually, a restarter will write a process contract for each new service it creates. Thus, events for each service will be sent individually to the restarter, where it can make decisions about how to respond to contract events and administrative requests. Each process can fail separately, but it is the restarter's job to decide how a process failure should affect the service.

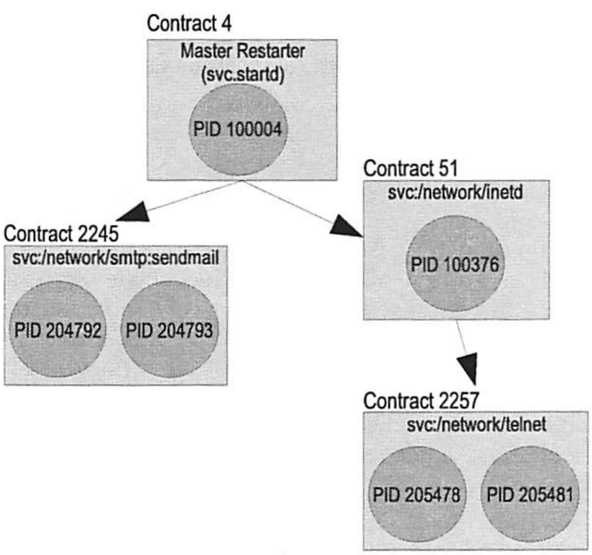


Figure 1: Contract model for software services.

A restarter determines how to interact with a service through its meta-configuration, which is stored in the SMF *repository*. The SMF repository stores persistent configuration and meta-configuration values as well as shorter-lived status information about each service. Each service has a common set of meta-configuration:

- The service name, in the form of a Fault Managed Resource Identifier (FMRI) is the unique identifier of the service on the system. The full FMRI for the sendmail service is: svc:/network/smtp: sendmail, but abbreviations like "sendmail" are available for interactive use.
- Service *dependencies* specify the relationships between services.
- *Method* specifications tell the restarter how to invoke the services, and may include invocations for start, stop, etc.
- The restarter defined by the service is responsible for starting, stopping, and restarting the service. Most services are managed by the default restarter, svc.startd. We've also implemented inetd as a service restarter.
- Services can optionally include localized, human readable descriptions and documentation references.

A service may also use the SMF repository to store simple configuration information. This allows application and system service developers to avoid maintaining configuration file parsers for small, simple configurations.

We developed a comprehensive state model for managing services, which are always in one of the following states: uninitialized, offline, online, degraded, disabled, and maintenance. These states have a consistent definition regardless of the service model.

State	Significance
Uninitialized	The initial state for all services. After evaluation by its restarter, the restarter will move the service to a maintenance, offline, or disabled state.
Disabled	The service has been disabled and is not running.
Offline	The service is enabled, but is not yet running. Often, a service will remain offline due to an unsatisfied dependency.
Online	The service is enabled, has started successfully and is currently running.
Degraded	The service is enabled, currently running but may be functioning at a limited capacity (e.g., reduced performance). The precise definition of degraded is service-specific.
Maintenance	The service is unavailable and cannot be automatically repaired by SMF. Administrative intervention is required to repair the service and clear the fault.

Table 1: State model for managing services.

Services transition between these states either because of administrative action (system startup, service enable, service disable), or service error (core dump, starting too rapidly, hardware fault). The service's ability to transition to a given state is always additionally influenced by the states of its dependencies.

To implement the service state model, SMF includes service restarters, which are themselves system

services (and hence subject to the state model). In our current work, a master restarter (svc.startd) and one delegated restarter (inetd) have been developed. While each of these are concerned with the restart of UNIX processes, the restarter architecture also allows implementation of non-process-based service models [6]. Each service restarter is responsible for defining appropriate transitions between the available SMF states.

Service Development

The meta-configuration and other simple service configuration are delivered onto the system by a small XML file, known as a service *manifest*. When the system starts up, the individual manifests are imported into the main SMF repository. The configuration of the system, including administrative customizations and run-time service data is stored in the repository and accessed by a common set of commands and APIs. The repository also provides the transactional semantics required for recovery on failure as well as the previous configuration snapshots which allow for configuration rollback.

In order to deliver an SMF-aware service, the administrator or software developer is required to create a service manifest. The manifest must include the meta-configuration for the service: a service name, the required methods, all dependency information, documentation references, and any other configuration to be stored in the repository.

SMF currently provides two restarters: svc.startd, the master restarter, and inetd, the inet service restarter. Both restarters require service definition through a manifest.

To provide a smooth upgrade path, service manifest creation can be an automatic step for inetd.conf services. We provide the inetconv utility to convert entries from inetd.conf to service manifests on operating system upgrade and allow administrators to convert entries subsequently added to the file. This is made easy by the well-specified service model for inetd services, and the fact that their dependencies are well known: inetd itself must be started, and rpc services require rpcbind.

Manifest creation for init.d scripts remains a manual process. The set of information required is well-defined, however two specific questions cannot be answered programmatically:

1) What are this service's dependencies? While we might be able to imagine a run-time checker or code analyzer for dependencies, these dependencies are often subtle and strongly tied to configuration. Automatic application dependency analysis would be a fascinating research topic of its own.

   The service author is usually best equipped to define the dependencies precisely. However, any consumer of the service can usually specify a dependency set that is sufficient for normal operation.

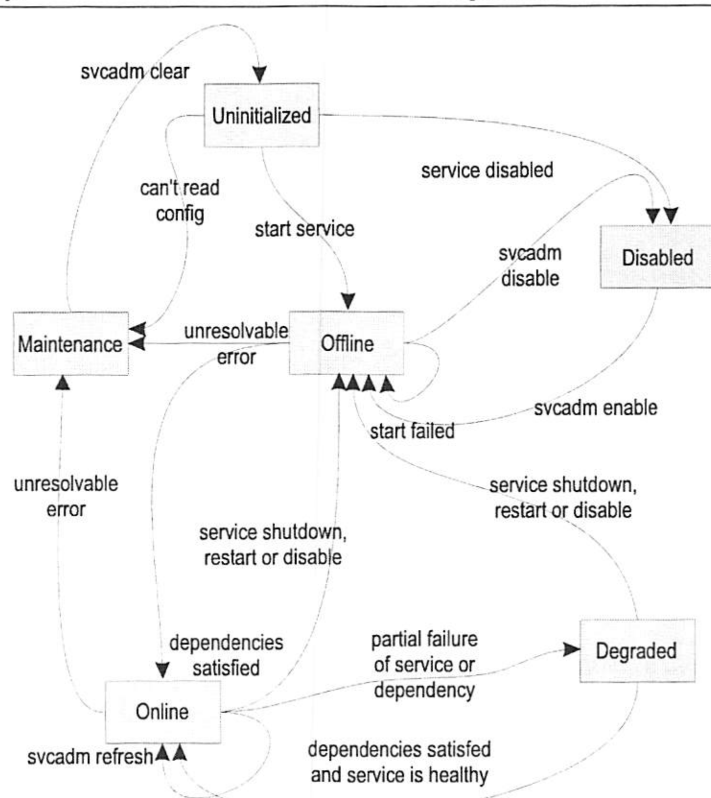


Figure 2: svc.startd state transitions.

2) What is this service's runtime behaviour? Does it have long-running processes that must be monitored, or is a lack of associated processes considered normal?

The service manifest author is required to specify dependencies in the manifest. Dependencies define service startup order as well as the restart relationships between services. A dependency is said to be satisfied if all conditions of its definition are met. There can be multiple dependencies for each service and each dependency may declare interest in multiple services. Two attributes of a dependency aside from the actual services in the dependency must be defined. The *grouping* specifies how the state of the services specified in the dependency are evaluated to determine whether a dependency is satisfied:

- *require_all*: all named services are running (online or degraded)
- *require_any*: at least one named service is running (online or degraded)
- *optional_all*: all named services must be either running (online or degraded), disabled, in maintenance, or not present on the system
- *exclude_all*: all named services must be either disabled, offline, in maintenance, or absent

The *restart_on* property determines which events should cause the service to be stopped:

- none: required only when the service is started

```
<service_bundle type='manifest' name='SUNWcsr:utmpd'>
<service
   name='system/utmp'
   type='service'
   version='1'>

   <create_default_instance enabled='true' />

   <single_instance/>

   <dependency
      name='milestone'
      grouping='require_all'
      restart_on='none'
      type='service'>
      <service_fmri value='svc:/milestone/sysconfig' />
   </dependency>

   <dependent
      name='utmpd_multi-user'
      grouping='optional_all'
      restart_on='none'>
      <service_fmri value='svc:/milestone/multi-user' />
   </dependent>

   <exec_method
      type='method'
      name='start'
      exec='/lib/svc/method/svc-utmpd'
      timeout_seconds='60' />

   <exec_method
      type='method'
      name='stop'
      exec=':kill'
      timeout_seconds='60' />

   <stability value='Unstable' />

   <template>
      <common_name>
         <loctext xml:lang='C'> utmpx monitoring
         </loctext>
      </common_name>
      <documentation>
         <manpage title='utmpd' section='1M' manpath='/usr/share/man' />
         <manpage title='utmpx' section='4' manpath='/usr/share/man' />
      </documentation>
   </template>
</service>
</service_bundle>
```

Figure 3: utmpd manifest.

- error: stop the service if the dependency fails due to hardware, software or administrative error
- restart: stop the service if the dependency stops for any reason
- refresh: stop the service if the dependency stops or is refreshed

Explicit dependency specification makes parallel service startup easy, regardless of whether the system is booting for the first time, or if a set of services had failed and are being restarted. Services are always started as soon as their dependencies are satisfied, so maximum parallelism for starting services is always achieved.

Unifying application deployment also provides a single location for access to advanced features. A service can be run with limited Privileges [11], or as an unprivileged or non-root user. Service management privileges may be given to authorized users without giving them full root access. A service may be bound to specific resource management limit and goal sets. All of this can be specified as service configuration with no code changes to the application itself (see Figure 4). Management of Solaris Zones [9] is simpler as SMF is available to each zone administrator.

Service Administration

The primary benefit a unified service management framework brings to system and service administration is a meaningful system-level view of all critical applications (see Figure 5). Services in an unexpected state are sorted at the bottom of the output for simple human consumption.

A primary goal of the SMF administrative model is to make common questions about services easy to answer and make common system administration tasks simple to perform. To evaluate the simplicity of the SMF model, the following provides examples of a few of these questions that were particularly difficult to answer in the init.d and inetd models prior to SMF.

What processes make up this service?

```
$ svcs -p sendmail
STATE    STIME   FMRI
online   May_06  svc:/network/smtp:sendmail
         May_06       9456 sendmail
         May_06       9458 sendmail
```

What's wrong with my system? (This will be covered in more detail in the next section.) See Figure 6.

What services does my service require in order to start? See Figure 7.

Which services won't be able to run if I disable this service? See Figure 8.

What services are available on this system? See Figure 9.

Messaging is also under the control of the administrator rather than the service. Messages previously emitted to console are stored in a per-service logfile, where they can be perused as part of post-mortem debugging or other diagnosis. They won't be lost to an insufficient terminal buffer size.

SMF defines a specific set of common administrative actions which can be applied to services:

```
# svccfg -s <svc> setprop start/user = astring: daemon
# svccfg -s <svc> setprop start/group = astring: daemon
# svcadm refresh <svc>
# svcadm restart <svc>
```

Figure 4: Configuring a service to run as "daemon" user and group.

```
$ svcs
STATE          STIME    FMRI
[...]
legacy_run     May_02   lrc:/etc/rc3_d/S81volmgt
legacy_run     May_02   lrc:/etc/rc3_d/S84appserv
legacy_run     May_02   lrc:/etc/rc3_d/S90samba
online         May_02   svc:/system/svc/restarter:default
online         May_02   svc:/network/pfil:default
online         May_02   svc:/network/loopback:default
online         May_02   svc:/system/filesystem/root:default
online         May_02   svc:/system/filesystem/usr:default
[...]
```

Figure 5: Abbreviated svcs output, including legacy services.

```
$ svcs -x
svc:/network/smtp:sendmail (sendmail SMTP mail transfer agent)
 State: maintenance since Tue May 10 18:35:41 2005
Reason: Method failed repeatedly.
   See: http://sun.com/msg/SMF-8000-8Q
   See: sendmail(1M)
   See: /var/svc/log/network-smtp:sendmail.log
Impact: This service is not running.
```

Figure 6: Checking the sendmail service.

- **enable** – Mark the service as enabled and start the service after all dependencies are satisfied.
- **disable** – Mark the service as disabled, stop the service, and do not allow it to start again.
- **refresh** – Reload service configuration and run the service's refresh method (if a refresh method is defined by the service).
- **restart** – Stop the service, then start it after its dependencies are satisfied.
- **clear** – Mark a service in the maintenance state as repaired, and if it is enabled allow it to start after all its dependencies are satisfied.

Administrators use svcadm to perform these administrative actions. Administrative intent is always preserved; disabling a service is guaranteed to persist across even patch and upgrade boundaries, where that was difficult in the past. The separation of administrative action and service state allows easier evaluation when system state doesn't match the administrative desire.

The use of a single API to take administrative action and change service configuration allows one point for security enforcement. Thus, SMF interoperates intimately with Role Based Access Control [11] mechanisms. It is easy to give a user just the ability to take action on a service (e.g., restart the service), but not change the service's configuration (see Figure 10). An administrator may also delegate authorizations with more granularity, giving privilege for an application

```
$ svcs -d sendmail
STATE  STIME   FMRI
online Aug_12 svc:/system/identity:domain
online Aug_12 svc:/system/filesystem/local:default
online Aug_12 svc:/network/service:default
online Aug_12 svc:/milestone/name-services:default
online Aug_12 svc:/system/filesystem/autofs:default
online Aug_12 svc:/system/system-log:default
```

Figure 7: Determining services needed by sendmail.

```
$ svcs -D system-log
STATE          STIME    FMRI
disabled       Aug_12   svc:/system/auditd:default
disabled       Aug_12   svc:/application/print/server:default
disabled       Aug_12   svc:/network/rarp:default
online         Aug_12   svc:/milestone/multi-user:default
online          5:55:34 svc:/network/smtp:sendmail
```

Figure 8: Determining services which require system-log's.

```
$ svcs -a
[...]
disabled       Aug_12   svc:/network/iscsi_initiator:default
disabled       Aug_12   svc:/system/metainit:default
disabled       Aug_12   svc:/network/ipfilter:default
disabled       Aug_12   svc:/network/rpc/nisplus:default
disabled       Aug_12   svc:/network/nis/server:default
disabled       Aug_12   svc:/network/ldap/client:default
[...]
```

Figure 9: Available service listing.

```
$ id
uid=37436(lianep) gid=10(staff)
$ grep lianep /etc/user_attr
lianep::::auths=solaris.smf.manage
$ svcs sendmail
STATE          STIME    FMRI
disabled       18:51:56 svc:/network/smtp:sendmail
$ svcadm enable sendmail
$ svcs sendmail
STATE          STIME    FMRI
online         18:52:43 svc:/network/smtp:sendmail
$ svcadm refresh sendmail
$ svcs sendmail
STATE          STIME    FMRI
online         18:52:55 svc:/network/smtp:sendmail
$ svccfg -s sendmail delpg autofs
svccfg: Permission denied.
```

Figure 10: Delegating service management authorizations.

administrator to only manage and change the configuration of a single service [10].

Finally, system security can be easily configured by creating a *profile*, which explicitly specifies which services are enabled and disabled on a system. To satisfy the common system configuration goal of no unencrypted network login services running, we provide the limited networking profile. Applying this profile explicitly disables all unencrypted network login services and enables other important services like ssh (see Figure 11).

Service Diagnosis and Self-Healing

svcs -x is a quantum leap forward in diagnosing problems with systems. It provides a very powerful paradigm. It describes what's going on in plain language with simplified output. It includes documentation references, as direct access to more information speeds the repair process. It points to an online knowledgebase; the website link included in the output contains the most up-to-date information on how to resolve the specific problem seen. Finally, svcs -x gives an assessment of the impact of each problems. If the specific issue effects no other services, that is stated explicitly. If other services are affected, svcs -xv will list them.

As the system (through dependency information) can determine the root cause of the problem, it can point the administrator directly to the component that must be repaired (see Figure 12).

In many cases, though, the administrator never needs to handle an error manually. Failure of a service due to hardware error, software bug, or administrative error can usually be resolved by restarting the service. This is handled automatically, with no administrative intervention. SMF also logs the error cause, to the extent it is known at the time of failure. This automatic recovery significantly reduces administrative costs.

To implement service restartability, SMF needed a way to detect when an error occurred in the service. Contracts provide a generic mechanism to express a relationship between a process and the kernel-managed resources it depends upon. The process contract allows

```
# svcs telnet
STATE          STIME    FMRI
online         17:49:15 svc:/network/telnet:default

# cd /var/svc/profile/
# cat generic_limited_net.xml
<service_bundle type='profile' name='generic_limited_net'
      xmlns:xi='http://www.w3.org/2003/XInclude' >
[...]
  <service name='network/ssh' version='1' type='service'>
    <instance name='default' enabled='true'/>
  </service>
[...]
  <service name='network/telnet' version='1' type='service'>
    <instance name='default' enabled='false'/>
  </service>
[...]
</service_bundle>

# svccfg apply generic_limited_net.xml
# svcs telnet
STATE          STIME    FMRI
disabled       17:49:40 svc:/network/telnet:default
```

Figure 11: Viewing and applying the limited network profile.

```
$ svcs -xv
svc:/system/filesystem/local:default (Local filesystem mounts)
 State: maintenance since Tue Sep 27 19:03:43 2005
Reason: Start method exited with $SMF_EXIT_ERR_FATAL.
   See: http://sun.com/msg/SMF-8000-KS
   See: /var/svc/log/system-filesystem-local:default.log
Impact: 23 dependent services are not running:
        svc:/system/sysidtool:net
        svc:/network/rpc/bind:default
        svc:/network/nfs/status:default
        svc:/network/nfs/nlockmgr:default
        svc:/network/nfs/client:default
        svc:/system/filesystem/autofs:default
        [...]
```

Figure 12: Service diagnosis of filesystem mount failure.

development of sophisticated restarters, which create a fault boundary around a set of processes, and receive and respond to events on processes within that boundary.

Implicit in service recovery is that the framework itself must also be fully restartable in the face of failures. An error of any user-land framework component all the way back to init can be caught, and the framework component itself will be restarted (see Figure 13). A transactional repository is required in order to implement the algorithms which recover from failure at any point.

Systems with sophisticated hardware error handling [7] make software recovery after error even more critical. Prior to SMF, the operating system developer when handling a hardware failure was given only a few unenviable options: kill the affected process and risk cascading failure, or restart the entire system. The operating system could not determine the broader effect of a faulted cell on a DIMM without inter-service dependency information. SMF manages error flow between services so that failures can be handled gracefully, by shutting down only affected processes and services which depend upon them.

Availability

The first version of the Service Management Facility is an integrated component of the Solaris 10 Operating System, released in January 2005. Themost recent copy of Solaris may be obtained free of charge at http://www.sun.com/software/solaris/ .

The Service Management Facility is also an integral part of OpenSolaris, which may be downloaded as source or binaries at http://opensolaris.org/os/ . The OpenSolaris SMF community contains information and discussion about service development and management in Solaris and OpenSolaris: http://opensolaris.org/os/community/smf/.

Experience

The first proof of concept and vetting of the design occurred when transitioning approximately 100 system services from their init.d script components into SMF. We wanted to confirm certain aspects of the SMF design, as well as take advantage of SMF benefits for Solaris service administration. The majority of services delivered as part of the Solaris operating system were migrated to SMF as part of its initial integration. We'll explore this case study here.

A small engineering team was responsible for creating nearly all of the 100 manifests for system services. A major lesson was that once a developer gains familiarity with the SMF model, manifest creation tends to take no more than a few hours per service, often including necessary testing. Experience with one manifest is directly leveraged for subsequent manifest creation. The fundamentals for basic service cooperation in SMF are relatively easy to grasp and do not seem to require significant training. The more sophisticated aspects of the service model tend to not be explored by the average service author, which we interpreted as success in our goal to make simple services easy to define.

As expected, the most challenging part of manifest creation was researching proper dependencies for each service. Sometimes the initial dependency analysis was incomplete – but, both point fixes for significant problems and longer term dependency additions to fill gaps were easy to perform.

We learned that someone familiar with the service implementation will be able to write a significantly better manifest than someone merely conversant in the service. A service manifest written by an end-user of the software is sufficient for that user's configuration, but not all potential uses and configuration of the service. Encouraging service authors to write

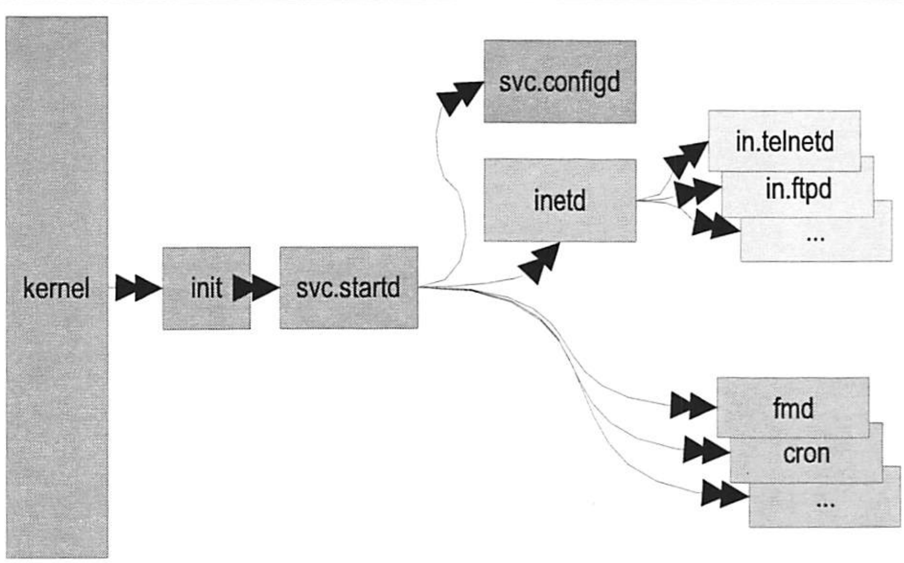


Figure 13: SMF framework restart relationships.

manifests for their software has significant value, even when a basic manifest has already been written.

Console interaction was more of a challenge than anticipated. When system startup was serial, console ownership was easy as it simply passed linearly from one script to another. SMF's parallel startup required tracking the console owner at all times, in case a system maintenance mode (also known as sulogin) was required to repair the system. The console ownership problem is a compelling reason for Solaris to provide access to the standard console login prompt as early in the boot sequence as possible. This convention is particularly helpful in a repair scenario because as much of the system as possible is available for use during the repair, rendering the repair environment less restrictive and more familiar.

Confirming the SMF design with critical system services started early in the boot sequence was likely a bit painful for the very early adopters. However, the lessons learned through actual service debugging informed significant usability enhancements which may not have been given the attention they deserved had SMF been an optional feature used only for a small subset of services.

Conclusions

A full-featured application and system service management infrastructure must reduce management complexity, increase application availability, and save administrators time. In order to provide all of these capabilities for a wide range of application models, the service must be elevated to a first-class administrative object which can be observed and managed. The Service Management Facility provides all of these benefits while requiring no changes to application binaries.

SMF reduces complexity by unifying and simplifying common administrative tasks across a broad set of applications. It increases application availability by detecting many critical errors and recovering from them automatically. Finally, it saves administrators time; when automatic recovery from failures is impossible, a complete management interface guides the administrator to the faulty component.

Author Biographies

Jonathan Adams joined Sun Microsystems, Inc. in Menlo Park four years ago, where he is a software developer in the Solaris Kernel Development group. His areas of expertise include memory allocation, inter-process communication, and debuggability. Reach him electronically at jonathan.adams@sun.com .

David Bustos graduated from the California Institute of Technology in 2002 with a BS in computer science. Since then he has worked in the Solaris operating system engineering group at Sun Microsystems, in Menlo Park, California.

Stephen Hahn is a Senior Staff Engineer in the Solaris Kernel Technologies group at Sun Microsystems. His recent work has been focused on service and resource management at the operating system level, particularly in building foundations for automated resource managers. His research interests are broad, and include describing meaningful application interdependencies, predictable systems behaviour, open source development processes, and implementing high performance sort algorithms. He received his Ph.D. in Theoretical Physics from Brown University, before joining Sun in 1997.

David Powell is a member of the Solaris kernel group at Sun Microsystems. In his six years at Sun, he has worked to improve the debuggability, availability, and approachability of Solaris, specifically focusing on inter-process communication and expressing dependencies between system resources and their consumers. He received his Sc.B. in Computer Science from Brown University

Liane Praza is a Staff Engineer in the Solaris Kernel Development group at Sun Microsystems. She's been at Sun since 1997, with areas of of expertise including the Solaris administration model, service and resource management, self-healing services, and clustered devices and filesystems. She received her B.S. in Computer Science from Purdue University. Reach her electronically at liane.praza@sun.com .

Acknowledgments

The SMF project was the work of a larger group of people than this paper represents, and we are grateful to everyone who worked to make this project possible. Dan Price, Dave Linder, and our USENIX shepherd, Tom Limoncelli, provided invaluable feedback and encouragement for this paper.

References

[1] Mewburn, Luke, "The Design and Implementation of the NetBSD rc.d system," *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, 2001.

[2] http://www.fastcoder.net/~thumper/software/sysadmin/chkconfig/ .

[3] Gooch, Richard, "Advanced Boot Scripts," *Proceedings of the Ottawa Linux Symposium*, June, 2002.

[4] http://developer.apple.com/documentation/MacOSX/Conceptual/BPSystemStartup/Articles/LaunchOnDemandDaemons.html .

[5] Bernstein, D. J., *Daemontools*, http://cr.yp.to/daemontools.html .

[6] Skinner, Glenn, et al., "A Service Management Facility for the Java Platform," *Proceedings of the 2005 IEEE Services Computing Conference*, 2005.

[7] Shapiro, Michael W., "Self-Healing in Modern Operating Systems," *ACM Queue*, Vol. 2, Num. 8, 2004.

[8] Candea, George, et al., "Recursive Restartability: Turning the Reboot Sledgehammer into a Scalpel," *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, 2001.

[9] Price, Daniel, et al., "Solaris Zones: Operating System Support for Consolidating Commercial Workloads," *Proceedings of the 18th Large Installation System Administration Conferences (LISA, '04)*, 2004.

[10] Brunette, Glenn, *Restricting Service Adminisration in the Solaris 10 Operating System*, http://www.sun.com/blueprints/0605/819-2887.pdf.

[11] Sun Microsystems, Inc., *System Administration Guide: Security Services*, http://docs.sun.com/app/docs/doc/816-4557.

Towards a Deep-Packet-Filter Toolkit for Securing Legacy Resources

James Deverick and Phil Kearns – The College of William and Mary

ABSTRACT

Users of a network system often require access to legacy resources. Providing this access is a difficult task for system administrators because the access protocols for those resources are typically insecure. A common approach is to develop a custom wrapper or proxy that securely processes user requests before forwarding them to the legacy server. The problem with this approach is that administrators must develop a custom solution for every resource. We believe that there are common requirements for managing these resources that can be addressed from a more centralized model. The userspace queuing extensions of the Netfilter firewall modules provide a generic environment in which protocol-aware deep packet filters can be constructed to enhance the security of resource access protocols. We employ this environment to strengthen two commonly used legacy protocols, and compare their requirements. We show that it is possible to secure legacy resources with minimal degradation in performance. We also discuss considerations for development of a deep packet filter toolkit to aid system administrators in securely managing legacy network resources.

Introduction

It has been our experience that, for a variety of reasons, production networks typically house several legacy resources. It may be the case that an application critical to the users of a network is no longer under active development, and no adequate substitute has been found. If a newer solution is available, other factors such as deployment cost or compatibility concerns may prevent its installation. The result is that system administrators must provide access to resources that employ inherently insecure protocols, a concept in conflict with their responsibility to maintain the security of the network and its resources.

By developing a way to manage these resources securely, administrators can continue to provide the needed services without endangering other components of the system or risking the integrity of the resources themselves. Currently, the only way to achieve this is to develop a custom environment in which to "wrap" each resource. No toolkit or framework exists that allows administrators to secure existing protocols with minimal effort. We believe that despite the arbitrary complexity of a protocol, certain common requirements exist for administrators that allow the development of such a toolkit.

As the basis for this toolkit, we employ the userspace queuing extensions of the Netfilter firewall [17] distributed with the Linux operating system. The processing capabilities provided by the userspace extensions allow us to collect any information deemed appropriate for a resource request before determining if the traffic should be allowed to contact the server. We can delay, alter, or reroute packets, interactively challenge authenticity, collect system status information, or anything else dictated by the protocol and administrative requirements of the resource at hand.

To illustrate both the power and flexibility of this approach, we consider two common protocols and the weaknesses they contain. In both cases, newer versions or extensions exist that address those problems, but we assume that some other consideration such as cost or compatibility prevents them from being deployed. This provides an example of what we believe to be a common issue in typical network system environments. Specifically, we discuss the lpr line printer protocol as defined by RFC 1179 [9] and the Network File System (NFS) protocol, version 3 [2]. Both of these systems provide essential resources to users, yet neither provides any strong authentication of those users.

Using the Netfilter userspace queuing extensions, we develop strong user authentication for each of these protocols. In the case of lpr, we employ public-key cryptography to ensure that users can only submit print jobs under their own identities. This prevents users from constructing raw print job control files in order to print under a false identity, possibly bypassing quota enforcement. Our technique also directly authenticates users, allowing administrators to open the print server to outside domains, such as wireless networks, providing users the ability to print securely from their laptops. For the NFS example, we use symmetric-key challenge-response authentication to secure filesystem mount requests. In order to mount a secured filesystem, a client must be in possession of the current key and know the format of the challenge issued by the firewall. This prevents unauthorized client machines from mounting remote filesystems simply by pretending to be authorized hosts.

Both of our extensions significantly improve the security of the original protocols, while introducing minimal overhead. They demonstrate how two vastly different protocols can be secured by the same basic technique,

and suggest a set of tools that could be used by administrators to secure legacy applications and protocols with significantly less effort than is currently required.

Design Goals

Faced with the problems of securing and managing legacy resources in a network, we define the following requirements for a useful solution:

- Neither client nor server components of any application should require modification to take advantage of the features we provide. Many existing approaches, such as application proxies, are not always feasible since they require application code to be rewritten or expanded.
- The solution must be generic enough to support any application with minimal overhead. Each application will impose different requirements on the solution. It should allow administrators to develop complex management systems without developing entire filtering applications from scratch.
- The system must be able to adapt to changing conditions in the environment, adjusting its filtering behavior accordingly. For example, depending upon application requirements, traffic from authenticated clients might not need to be filtered for a given period of time.

General Related Work

Because we deal with protocol-specific traffic filtering, the most relevant existing work relates to *deep packet filtering* [7]. Several authors have acknowledged that the basic address/protocol-based filtering provided by traditional firewalls is not sufficient for many applications. In cases where filtering requirements are tied directly to a given protocol, stock firewalls are not appropriate. Instead, a filter that examines protocol-specific information contained in the packet payloads provides the required service. Because deep packet filtering incurs a higher cost than traditional header-based filtering, much of the research attention has been paid to improving performance of these filters, even via hardware solutions [4, 6, 18]. Very recently, it has been suggested [5] that forthcoming firewalls must include some level of deep packet filtering capability, due to the increasing number of application-level attacks. We agree, and the toolkit we develop is an essential step toward integrating these protocol-aware filtering capabilities in commonly existing software-based firewalls.

Implementations

Here we describe two prototypical applications chosen to demonstrate the capabilities of our technique. Each is exemplary of different components of the system interacting with different protocols. Both of them were implemented on a six machine testbed in a secure laboratory [12]. Figure 1 illustrates the testbed layout. The firewall separates two private subnets. On one subnet resides three test clients that interact with a server residing on the other subnet. They are connected by a path that travels through the firewall. This is designed to emulate a typical setup in which a perimeter firewall protects a server or set of servers from client traffic, whether it originates from an internal network or an outside internet. We also place a client-class machine on the server subnet to provide control data for performance analysis. Since it is on the same subnet, the control client need not traverse the firewall to interact with the server. The machine is composed of the same hardware that constitutes the other three clients. By running identical client-server applications on the control client and the firewalled client, we can derive the overhead imposed by our toolkit. Note that a second path exists through which traffic could possibly travel from clients to servers, but this connection is not a part of the testbed. It is used for administrative purposes in the lab environment.

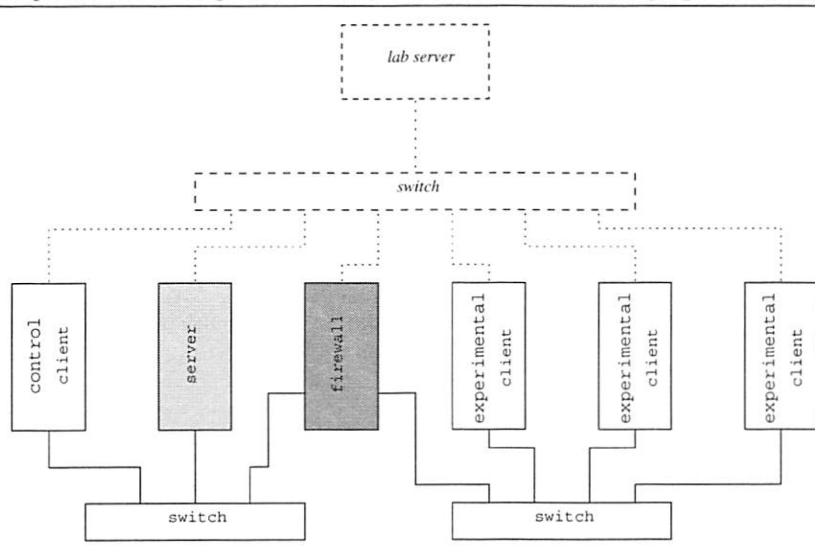


Figure 1: Prototype testbed.

iptables has a modular extension interface that allows developers to introduce additional features. One module included in the distribution introduces a QUEUE target that stores packets in a kernel data structure until they are retrieved by a userspace daemon. This daemon can then analyze the packet and its contents, and determine an appropriate course of action using the resources provided to userspace programs. Figure 2 illustrates the components of a firewall system that uses this approach. Since it allows us to implement more detailed packet analysis than the built-in features of iptables, we employ this module in the implementation of our deep packet filter daemons.

NFS Prototype Implementation

The Network File System [2] allows filesystems on a sever to be exported over a network to multiple clients. Clients can mount the filesystem, which then appears as part of the local directory structure; network operations are transparent to the user as he interacts with the remote files.

Since the filesystems being exported to remote clients may contain sensitive information, we must ensure that only authorized clients are able to mount them. The stock implementations of NFS provide simple host-based authentication by comparing the IP address of a client from which a mount request originates against a table of pre-authorized client addresses. To further increase security, some environments employ client MAC address filtering at the network switch level.

Even in this case, however, a malicious user can gain unauthorized access to the filesystem. He needs only to set manually the MAC address for his network interface, configure his client with an authorized IP address, and unplug a legitimate system. He can then connect to the switch, replacing the connection of the system whose MAC and IP addresses are being spoofed. By taking these steps, the user presents a client to the system that, in most configurations of the currently prevalent protocols, is indistinguishable from a valid, authorized client. These considerations dictate the need for stronger authentication of clients wishing to mount remote filesystems.

Related Work

Despite the presence of new, more secure, file sharing paradigms [13, 3], traditional NFS is still widely used. Accordingly, some attempts have been made to strengthen authentication in the NFS protocol itself. Ashley, et al. [1] introduced role based access control to the protocol. O'Shanahan [15] replaced the UID-based authentication in NFS with a public-key cryptosystem. A less intrusive modification was proposed by Goh, et al. [8]. In their system, the client intercepts all filesystem operations and encrypts them on the fly so that they are secure when stored on the server. The underlying NFS structure isn't changed,

server

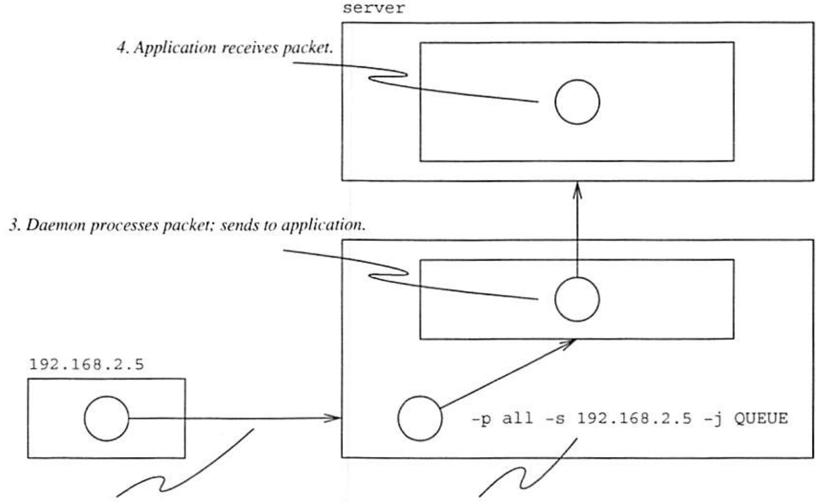


Figure 2: Packet filtering with userspace queues.

```
IPTABLES -A FORWARD -p TCP -s $CLNTNET --dport sunrpc --tcp-flags syn syn -j ACCEPT
IPTABLES -A FORWARD -p TCP -s $CLNTNET --dport nfs --syn -j ACCEPT
IPTABLES -A FORWARD -p UDP -s $CLNTNET --dport nfs -j ACCEPT
IPTABLES -A FORWARD -p UDP -s $SERVER --sport nfs -j ACCEPT

IPTABLES -A FORWARD -p TCP -s $SERVER --tcp-flags ack ack -j QUEUE
IPTABLES -A FORWARD -p TCP -s $CLNTNET --dport sunrpc --tcp-flags ack ack -j QUEUE
IPTABLES -A FORWARD -p TCP -s $CLNTNET --dport sunrpc --tcp-flags urg urg -j QUEUE
IPTABLES -A FORWARD -p TCP -s $CLNTNET --dport sunrpc --tcp-flags fin fin -j ACCEPT
IPTABLES -A FORWARD -p UDP -s $CLNTNET --dport sunrpc -j QUEUE
```

Figure 3: Initial ruleset for authenticated NFS.

but the data being stored within that structure are modified transparently to the user. Our approach is different in that no components of the system require modification of any kind. Server and client daemon behavior is unchanged, as are the data and metadata stored in the remote filesystem. The only component we introduce is a client-side authentication daemon that responds to challenges issued from the firewall during mount attempts. Assuming that clients are not able to bypass the mount process, we gain a more secure NFS environment. If, however, a client successfully guesses a remote filehandle, it can modify the associated file by manually constructing NFS packets and interacting directly with the server daemon. Our approach, like any system that does not authenticate every client-server interaction, cannot prevent this form of compromise.

Connection Life Cycle

In order to challenge traffic destined for NFS mount daemons, we must first learn the port at which those daemons are running. Since the portmapper query for this information will always be destined for the sunrpc port of the server, we can easily intercept this traffic, remember that a client has requested this port information, then allow the request to proceed.

Dynamic Rules

Our obligation to intercept any portmapper request defines the initial iptables ruleset, illustrated in Figure 3.

When the server responds, we also intercept that information before forwarding it to the client, and extract from the reply the ports on which client's mount request will be made. We then inject a new rule into the firewall that QUEUEs the client's forthcoming mount request to the authentication daemon. The system can handle requests from multiple clients simultaneously, provided that the Netlink queue buffers do not overflow. If this occurs, packets are dropped as though the firewall discarded them, or network congestion occurred.

Authentication Mechanism

We include an encrypted challenge-response scheme to authenticate remote clients. When a client issues a mount request, it is intercepted by the userspace daemon on the firewall and queued in a data structure to be either accepted or dropped depending on the result of the authentication process. The filtering daemon maintains a state variable for each packet that records that packet's sequential arrival order at the firewall. Since the value of this variable is effectively unique for each packet over the lifetime of the system, we can use it to key a data structure that records all unanswered challenges. Our prototype uses a linked list to store this information, allowing multiple simultaneous client connections. Figure 4 illustrates the structure of a single list node.

```
struct list_element
{
    ipq_packet_msg_t *packet;
    long packet_id;
    long nonce;
    struct list_element *prev;
    struct list_element *next;
};
```

Figure 4: NFS packet queuing structure.

The userspace daemon issues a challenge to the client consisting of a symmetrically encrypted id-nonce pair, and waits in the background for additional traffic to arrive. This traffic may be another mount request that triggers the same process, lengthening the queue, or it may be a client's response, determining the fate of the mount request. The daemon can apply

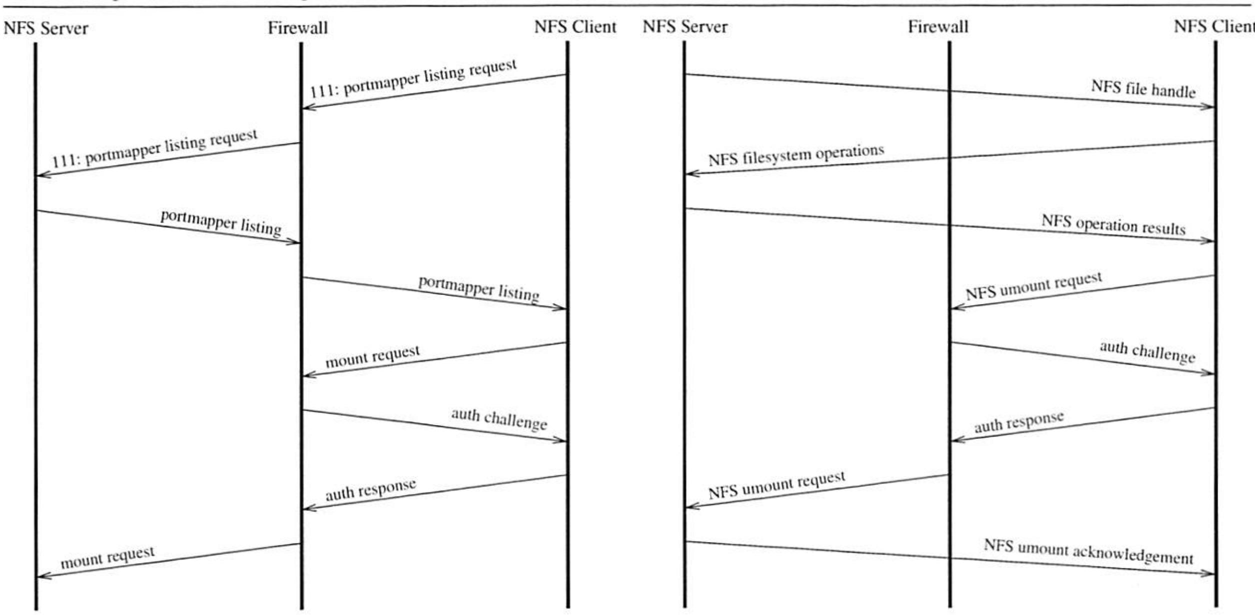


Figure 5: Authenticated NFS service.

to a packet the same primitive filtering targets as the kernel firewall. A library interface provided by the queueing extension module described earlier provides these targets, allowing the daemon to ACCEPT or DROP a packet. If the client's response is satisfactory, an ACCEPT verdict is issued on the original packet, and the mount request is processed by the server. If not, then the packet is dropped, and the server never sees the request. With encryption keys pre-loaded on systems and restricted to root access, a malicious user must successfully root-compromise a legitimate client to subvert the protocol. Spoofing network addresses of any type is no longer sufficient to gain access to the remote filesystem.

Assuming that authentication succeeds, the firewall simply forwards all other NFS traffic between these two systems; no further intervention is imposed until the session ends. Since unmount requests are also processed by mountd, an authentication challenge will be issued at this stage as well. This is a desirable behavior, as it prevents an unauthorized user from unmounting a filesystem from beneath a legitimate user. Figure 5 illustrates the message passing in the authenticated system.

Host Failures

In the event that the server crashes, the client loses connectivity with the remote file system, but only while the server is down. Since authentication only happens during mounts, a reboot of the server simply delays NFS calls until the reboot is complete. The client does not have to remount the filesystem unless something catastrophic enough to change the filehandle on the server side has occurred.

Should the client crash, all mount information is lost during the reboot. the client will need to remount the remote filesystem, requiring that the authentication process be repeated. The firewall and server have no way of knowing that the client crashed, and assuming that the mount request is legitimate would open the system to attack.

Performance

To measure the performance overhead associated with the system, we run benchmarks from two client machines and determine achieved bandwidth in the filesystem. One client mounts the NFS share through the firewall, while the control client resides on the same subnet and has a locally switched connection to the server. Figure 1 illustrates the distinction.

Measurements were taken using the IOZone Filesystem benchmark suite [10]. First, a 32 MB file is written to the NFS share. The suite records the amount of time required to write the file and calculates the achieved bandwidth. A similar approach is used to measure reading performance. We present benchmark results for sequential and random writing and reading of records ranging from 2KB to 1024KB in a randomly generated 32 MB file. Data are presented within 95% confidence intervals generated from 30 independent measurements of the system's performance.

The benchmark suite allows us to compensate for many caching effects. We employ some of these features to present clearer results. Between every test, the filesystem is unmounted; this clears the filesystem buffer cache, and forces the next test to interact directly with the filesystem instead of a faster cached copy. We also employ a setting that flushes the processor caches between each test to ensure that no data are cached locally during read tests.

It is important to note that the benchmarks measure achieved bandwidth in the filesystem only after it is mounted. Time required to mount and unmount the filesystem is not included the results. This means that the authentication mechanisms used must be timed separately. What we measure here is how much the presence of the firewall between the client and server slows down normal operations.

First, we consider the slowdown imposed by the firewall on sequential read and write operations. Figures 6 and 7 illustrate the performance difference between the firewalled and control clients. In both cases, the slowdown is likely to be less than 1% for any record size. We consider this a negligible amount of overhead.

Next, we examine the random read and write cases. Figure 8 illustrates another negligible slowdown. In this case, the worst slowdown is just under 3%, with most cases falling well below 1%. The only case in which we see a significant slowdown imposed by the firewall is randomly reading records from the file, as illustrated in Figure 9. Here, the extreme case imposes a slowdown of about 11%, with most cases incurring a slowdown near 6%. Given that only one type of operation incurs a significant slowdown, and that slowdown, on average, is only about 6%, we argue that the overhead imposed by the system is small enough to justify the additional security afforded by our approach.

The time required to mount the remote filesystem, including the encrypted authentication exchange, was 268 ± 11 µS on the firewalled client. The control client, which has a direct connection to the server and does not perform an authentication exchange, mounted the filesystem in 16 ± 0.2 µS. Relative to the performance of the control client, the firewall introduces significant overhead in mount the remote filesystem. Note, however, that this overhead is encountered only when the filesystem is mounted, and the real time required to mount the directory is still extremely small. Amortized over the lifetime of the directory mount, the overhead is insignificant.

LPR Prototype Implementation

RFC 1179 [9] defines the widely used lpr printing protocol. It contains no support for secure authentication of users. In environments where printing is monitored

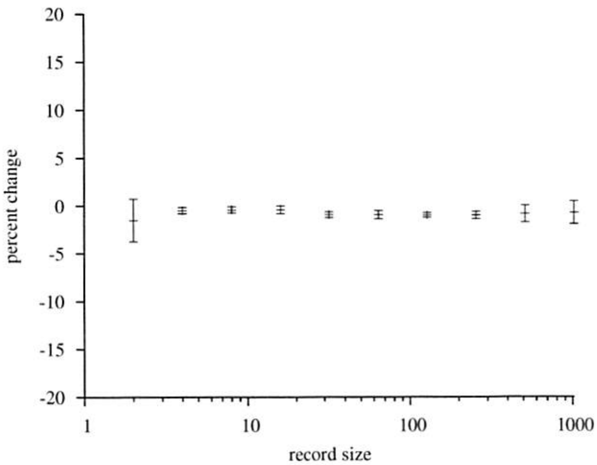


Figure 6: Firewall performance relative to control client. Sequential read of 32MB file. 95% confidence intervals.

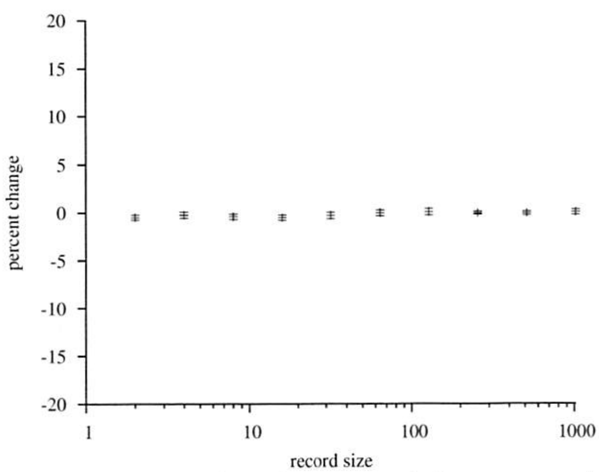


Figure 7: Firewall performance relative to control client. Sequential write of 32MB file. 95% confidence intervals.

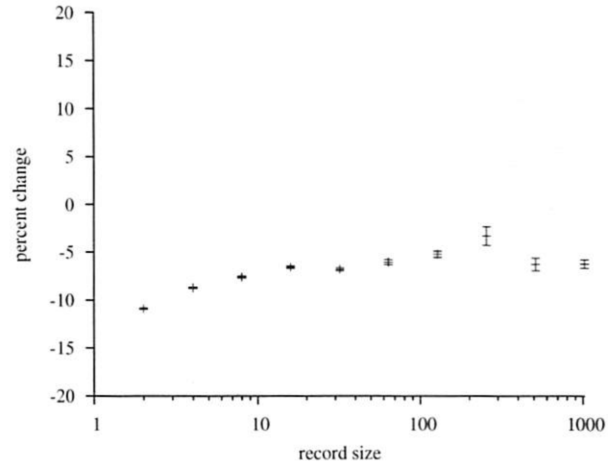


Figure 8: Firewall performance relative to control client. Random read of 32MB file. 95% confidence intervals.

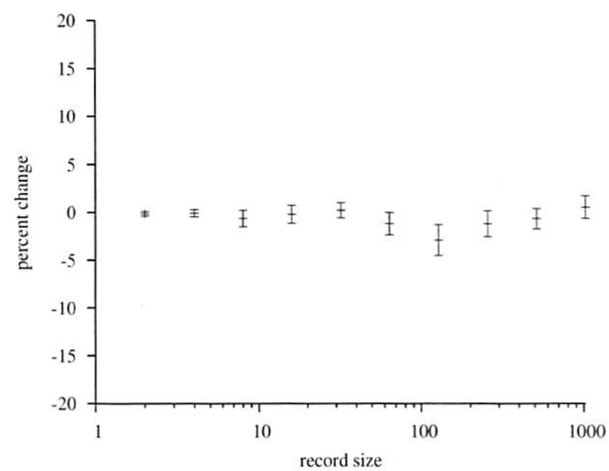


Figure 9: Firewall performance relative to control client. Random write of 32MB file. 95% confidence intervals.

monitored or under quota enforcement, this issue is significant. It means that a user can easily submit printing jobs under another user's ID, thereby bypassing the cost associated with his job request. This same issue complicates the matter of allowing authorized users to print without restriction based on network topology. Ideally, an associate of a department should be able to print to that department's printers whether using his desktop workstation or his laptop from home. Since, however, the protocol doesn't support strong authentication of users, allowing such open access would certainly result in numerous unauthorized printing requests.

Related Work

Many approaches exist to solve this problem. LPRng [16] introduced a new implementation of the protocol that supports the RFC 1179 interface while providing additional functionality such as Kerberos authentication [14] and public-key cryptography support. Most secure printing protocols that conform to the lpr specifications are extensions of LPRng. While this approach does solve the authentication problems we outline, LPRng is only compatible with UNIX-like operating environments. Windows clients, while capable of printing to RFC 1179 lpr servers, do not understand the extensions in LPRng. Accordingly, they cannot take advantage of the authentication attributes provided by the implementation. We offer an enhancement to the standard RFC 1179 protocol that requires no changes on either the client or server side, yet still introduces a strong authentication scheme.

A common approach to strengthening the security of a legacy protocol is to tunnel all of its traffic through an encrypted channel, usually with SSH. This does provide stronger security, but the degree of improvement is highly application specific. In this case, it only partially authenticates the user. An SSH tunnel authenticates the connection, but not the use of

the protocol at hand. Suppose a legacy line printer system is protected by an SSH tunnel, through which end users must forward jobs.

The result is that only authorized users will be able to submit jobs, but there is no enforcement that those jobs be submitted in the correct name. A user may have an account on the system, allowing her to establish an SSH tunnel to the print server, but nothing stops her from submitting the job under false identification (as described below) once that tunnel is in place. A similar property holds true for any encryption or tunneling scheme, including virtual private networks or IPSec tunnels. If the underlying protocol doesn't directly support user-specific strong authentication, then secure channels do not provide robust authentication in typical production environments.

RFC 1179 requires that clients submit a print job control file to the server in order to process a new job. Contained in this control file is the username of the entity submitting the job. The problem is that this field in the control file is presented in cleartext. Nothing prevents a malicious user from altering the contents of this file before the job is submitted, thereby tricking the server into printing the job under someone else's userid. The scheme that we offer forces the user to demonstrate with very high probability that he or she is, in fact, the user designated in the control file for the job being submitted.

Simply put, we intercept the TCP syn packet associated with the connection to lpd, the print server process, and delay it at the firewall until we receive a digitally signed message from the client informing us of the owner of the incoming job. Once this information is stored, the TCP connection is allowed to proceed, and when the control file begins to arrive at the firewall, the daemon extracts the job user information from the file and compares it to the signed username previously received. If they match, then we know that the user for this job is authentic, so we allow all other traffic on this connection through. If not, then we immediately tear down the connection. The server will abort the job, and the client's only option is to restart the job submission process.

User Authentication in Advance

The initial ruleset for our approach is as illustrated in Figure 10.

This small ruleset simply propagates all lpd-bound packets to our userspace daemon for processing, including the TCP packets used to construct the connection over which the job transfer will take place.

A client sends a print job to the print server in the standard fashion. Between the client and the print server is the firewall, listening for connection attempts destined for the print server's spooling ports. When a TCP connection request is made, the firewall intercepts

```
IPTABLES -A FORWARD -p TCP -d $SERVER --dport printer -j QUEUE
IPTABLES -A FORWARD -p TCP -s $SERVER -j ACCEPT
```

Figure 10: Initial ruleset for authenticated remote printing.

```
struct client_element
{
    // together these should indicate a TCP connection
    __u32                   sourceip;           //source address of client
    __u16                   sourceport;         //origin port on client

    ipq_packet_msg_t        *syn;               //syn packet for connection
                                                //will challenge this

    // timeouts are needed to allow reuse of ports
    struct timeval          lasttime;           //last activity time

    // authentication attributes
    long                    nonce, id;          //to prevent replay attacks
    unsigned char           user[32];           //username from challenge
                                                //control file must match

    int                     checked;

    // RFC 1179 defines the command protocol
    unsigned char           current_command;  //last level 1 cmd rec'd

    // control file is subcommand 02 after "receive job"
    unsigned char           current_subcommand; //last level 2 cmd rec'd

    long                    ctl_length, ctl_consumed;     //ctl file length
    long                    data_length, data_consumed;   //data file length

    int                     fin;

    // list operators
    struct client_element   *prev, *next;
};
```

Figure 11: lpr packet queuing structure.

the syn packet and stores it in a data structure until it can confirm the identity of the client. Figure 11 illustrates the data structure used to store client information.

After storing the TCP syn packet, the firewall issues a challenge to the client machine. A specialized daemon on the client listens for and responds to these challenges. The format of the challenge is simple; it consists of a packet id and a random nonce to avoid replay attacks. When the client-side daemon receives this challenge it constructs a response consisting of the same packet id, the incremented nonce, the userid of the owner of the daemon (presumably the person submitting the job), and the digital signature for the message as associated with the given userid. Figure 12 illustrates the response issued from the client upon receiving a challenge from the firewall.

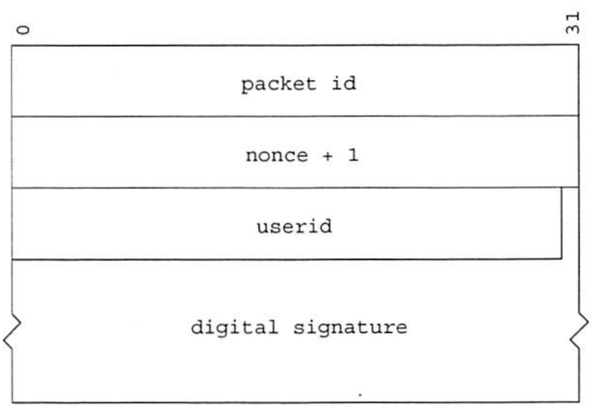


Figure 12: Structure of the printing challenge response.

When the firewall daemon receives the client's response, it first verifies that the signature on the response is valid. To accomplish this step, enterprise environments will require the employment of a scalable public key infrastructure. The details of of such a system are beyond the scope of our discussion. The prototype we present simply assumes that the public keys for legitimate users are located in a directory to which the firewall daemon has read access. The daemon searches that directory for a key with the name <userid>.pem, where <userid> is extracted from the cleartext portion of the client's response. If the signature is not valid for the message, then we do not allow the TCP connection request to reach the server at all.

To enhance the performance of the system and reduce unnecessary network traffic, we do not free the allocated data structures for the connection right away. Because the daemon must only assume a strict conformity to RFC 1179, we do not know what to expect from the client in response to the initial connection packet being dropped. LPRng version 3.8.19 will issue three connection attempts, each timing out at 10 seconds, before giving up on the connection. Alternate implementations of the protocol may use different approaches. Some may even depend solely on the reliability components of TCP to establish the connection. Once the data structures are de-allocated, syn packet retransmissions from the client will be seen by the firewall as new connection attempts, resulting in a repetition of the authentication process. We avoid this by leaving the data structures in place long enough to ensure that the syn packets are re-transmissions and not new connection requests; having this information allows us to silently discard those packets without consuming additional bandwidth and processor time. In the prototype implementation, this timeout variable is static. For a production release, we would allow a configurable runtime option to set this variable, since the administrator would have the context information needed to optimize it.

At this point, the firewall has created an instance of the data structure illustrated in Figure 11 and loaded it with with client IP address/port number combination and the verified username of the entity submitting the request. By indexing the data structure on host IP/port pairs, we can handle inbound packets from a number of different connections simultaneously.

Administrative Jobs

The LPR protocol allows the superuser significantly more control over job submission and administration than regular users. For example, an administrative user can issue or remove a print job with a manually specified username as the owner of that request. The benefit of this is that system administrators can delete pending print jobs through the LPR interface without having to assume the identity of the actual owner or manually modify the state of the printing system.

This means that for connections submitted by the root user of a client machine, the embedded user information may not match the credentials presented during authentication of the connection. Suppose that Mary needs to print a large document, and the department in which she works has a specially designated printer for large jobs. She sends the job to the print server without specifying the correct printer, and the job lands in the queue for the default printer, which can't correctly handle the large job. The system administrator notices the problematic job, and sends an lprm command for that job to the print server as the root user on Mary's machine. The server deletes the pending job, and Mary can be informed that her job was canceled and she should submit it to the correct printer.

If we absolutely require that requests be submitted in the name of the user whose credentials were presented during authentication, we break this component of the protocol. Accordingly, we must make a special case for jobs submitted from root users. If the challenge response packet contains root as the userid, and is properly signed, then we can make no assumptions about what user information the actual job request will contain. With a valid signature on the

challenge response, we know that the client must be an authorized administrative user. Therefore, we place no restrictions on what can be done over the lifetime of that connection.

lpr Job Interception

Once the authentication process is complete (and assuming the authenticated user is not root), the userspace daemon still intercepts the traffic, because it must confirm that job is actually submitted as the user who authenticated with the firewall. It observes the stream as it passes through, watching for the octet that indicates the user field of the control file. The control file consists of commands, subcommands, and data, exactly one combination of which indicates the user for the current job. Specifically, we let everything through until we reach command 2 (receive job), subcommand 2 (receive control file), at which point we begin parsing the control file.

Each line in the control file has a specific format: some character indicates what data will appear on the line, then the rest of the line is that data. For our purposes, we skip lines in the control file until we see one that beings with the character 0x0A, which indicates the ascii username of the owner of the job being submitted. If this user matches the user that previously authenticated, we forward all remaining traffic to the server. Because there may be additional packets already queued in the Netlink buffers, we leave the userspace data structures in place and set a flag indicating that everything else on that connection should immediately be forwarded.

To enhance performance, we also inject two rules into the kernel firewall tables. One forwards everything except the TCP fin packet for the connection directly to the server. By bypassing the userspace propagation for the remaining packets, we significantly reduce the amortized cost of the authentication process. The second rule matches the TCP fin packet from the client, indicating the end of the connection. This packet is forwarded to the userspace daemon, which de-allocates the data structures for this client, then forwards the final packet to the server.

If the user in the control file does not match the authenticated user, no further data will be sent to the print server, and the connection between the server and client will be aborted. Since the job is not actually processed until all of the control file is received, the document will never print.

Authentication Failure and Cleanup

When the firewall denies a print job that has failed the authentication process, the server already has begun receiving the print job. Once the job is denied, these connections must be destroyed so that the server can process additional jobs in its queue. If we do nothing, eventually, both the server and client will timeout and the job will be aborted. The problem with this approach is that in the meantime, jobs accumulate in the queue and are delayed while the server waits for additional data that will never arrive. A better approach is to actively destroy the connection, resulting in an immediate abortion of the job on the server side. We employ the techniques of Lowth [11] to accomplish this. Excerpts from his TCP/IP connection cutter software, available under GNU Public License, satisfy our requirements that both ends of the connection be shut down upon seeing an invalid user for a print job.

Performance Analysis

We measure the impact the filter has on two aspects of the printing system: connections and bandwidth. We present these data separately because there is a significant amount of initial overhead on a per-connection basis due to the asymmetric encryption used for advance user authentication. Once that is complete, the filter imposes minimal overhead on data transmission. Measurements were conducted from both firewalled and control clients, as in the NFS benchmarks.

To calculate job submission times, we modified the source code of the print server daemon to record the arrival time of the first and last packets on the connection, and present the difference in a log file. The same file was printed from both the firewalled client and the control client in these measurements. Submitting a 17 byte text file from the control client took $47{,}719 \pm 3,569$ μS. As expected, the firewalled client achieved a slightly slower rate of transmission, requiring $202{,}420 \pm 21,660$ μS to complete.

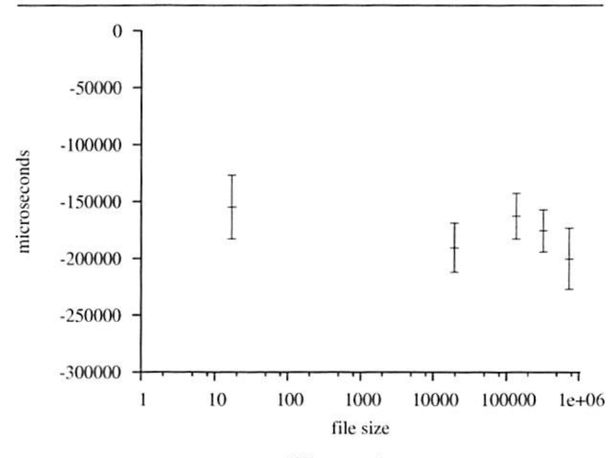


Figure 13: Firewall performance relative to control client. Receipt of control and data files at print server. 95% confidence intervals.

Unlike connection times, which experience a small, static delay with respect to file size, we see in Figure 13 the impact of channelling each packet through the firewall. As files grow in size, more packets are required to contain them; each of these packets must traverse the firewall, experiencing a small delay that is cumulative for submitted job.

Future Work

Having constructed prototypes and described potential uses of our system, we now outline specific tasks that we intend to pursue in the next stages of the project. We present our envisioned final product, complete with additional examples of applications that can take advantage of the services it provides. We are currently developing a firewall management toolkit for the Netfilter system that uses the userspace queuing extensions to allow more active participation from the firewall in the system at large. Figure 14 illustrates the proposed basic structure of the system. The toolkit will include several built-in functions to achieve tasks that we believe will be common in applications. Some functions will be statically implemented; others will be implemented via a pluggable module interface. This allows implementations to be swapped out as appropriate for specific applications.

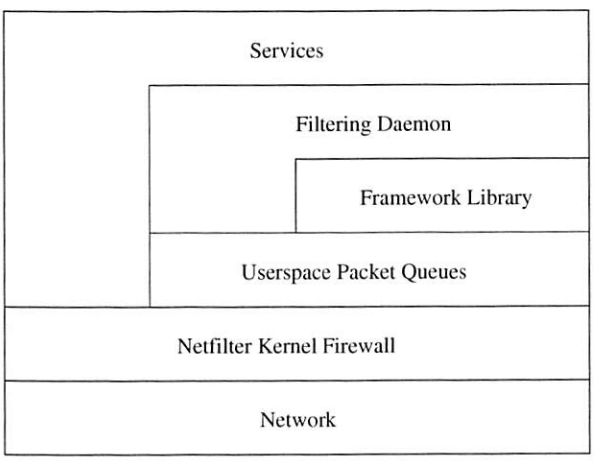


Figure 14: Basic toolkit structure.

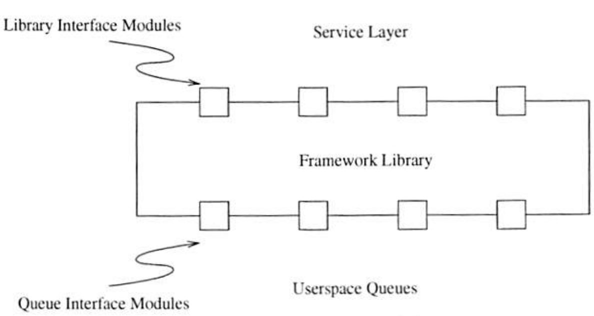


Figure 15: Toolkit modules.

Illustrated in Figure 15, we plan to include a module interface at both ends of the toolkit in the interest of keeping the solution generic. The manner in which the system interacts with filtering daemons is defined by the library interface modules. Similarly, the system's interaction with the Netfilter queuing system is defined by the queue-level modules, though the variability in the latter should be less prominent given the restrictive interface of the queuing system.

Note that services need not be placed behind protections provided by the firewall toolkit. They still have access to basic services provided by the underlying Netfilter system, should the administrators deem it appropriate. Our proposed toolkit is an extension of the firewall model, not a replacement for it. The administrator can construct rulesets that direct traffic for only some services through the system. Other services can use traditional protections, since only those rules that specify the QUEUE target will be propagated to the toolkit.

Planned Toolkit Modules

The prototypes we have implemented dictate a set of useful modules for the toolkit. They provide basic functionality that we believe will be common in several filtering applications. Here, we outline several of the planned toolkit modules. The complete list of included modules will be determined as we construct the system and determine common needs among potential applications. A systems administrator can implement and supply additional modules as deemed necessary for the desired application. This shields application development from cryptographic implementations and allows the administrator to focus on more abstract functions appropriate to the application at hand.

Cryptographic Support for Authentication

Since nearly every application we have proposed relies heavily on authentication and authorization, we must provide direct support for those features in our toolkit. Different forms of authentication require different types of cryptography, so the interface should be generic and configurable, providing support for arbitrary cryptographic modules.

Common paradigms such as RSA digital signatures and fast Blowfish encryption will be included toolkit modules in the system we build. Instead of relying on high-level generic interfaces provided by the standard cryptographic libraries, the administrator will have access to a suite of cryptographic functions tailored to packet-level encryption and challenge-response authentication schemes.

Connection Destruction

In most cases where an authentication challenge fails, we envision the need to abort any pending connections to the server. In the NFS case, this is trivial, since authentication failure is detectable before a client actually connects to the server, and subsequent application traffic is sent over the connectionless UDP protocol. A failed authentication simply means that the firewall blocks all application traffic from the suspicious client, and none reaches the server.

The lpr example demonstrates a more difficult case, however, where authentication cannot be detected until *after* the server connection is in place. Here, we must forcefully terminate the open connection to prevent the

client from sending any additional data to the server. The prototype includes code to accomplish this based on [11], but it adds unnecessary complexity to the filtering daemon. Such a common function is exemplary of the modules our toolkit will include.

Ruleset Modifications

Both of the prototypes we preset modify the kernel firewall's ruleset over the lifetime of the system. The NFS example injects a rule to QUEUE incoming mount requests after it learns the port location of the file server's mount daemon. The LPR prototype injects a rule to accept all remaining traffic on a connection if authentication succeeds. The first case is functional in nature; the second is a performance enhancement.

Both implementations achieve this via invocations of the userspace iptables utility. Since that utility interacts with the kernel firewall through system calls that modify network sockets, we can achieve the same functionality directly. We will include a module that provides a library interface to the underlying firewall ruleset, allowing direct modification of the rules in response to changing system conditions. This will significantly reduce the overhead required to provide a dynamic ruleset.

Implementations and Evaluations

With a toolkit in place, we intend to revisit the two prototypes presented earlier, and re-implement them using the firewall toolkit. This will allow us to offer a comparison of brute force implementations with toolkit assisted constructions. We can compare the ease and efficiency of implementations, and analyze the differences in performance overhead obtained in each method.

Because we are dealing with a real-time environment, performance is of paramount concern. We must take steps to ensure that our toolkit introduces minimal overhead. In order to achieve this, careful measurements must be taken on each module that we introduce.

Conclusions

We have presented two prototypical applications that illustrate how the userspace queuing extensions of a commonly available firewall can be used to secure legacy protocols. The performance imposition of our system is minimal, and certainly justified in light of the increased security and functionality the system can introduce. We have discussed how this approach can be extended into a toolkit that system administrators can use to secure additional legacy protocols and introduce additional arbitrary functionality at the firewall. We believe that the toolkit we discuss will allow administrators to balance more easily the need to provide existing services to users with the need to maintain security features in a networked environment. Finally, we have outlined a larger, more general application to which our toolkit will be useful, demonstrating the flexibility of our approach.

Author Biographies

James Deverick is currently a Ph.D. candidate and systems administrator at The College of William and Mary's Department of Computer Science. His research focuses on building active firewalls that enhance the security of legacy systems. Reach him at jwdeve@cs.wm.edu .

Phil Kearns is an Associate Professor of Computer Science at the College of William and Mary. His research interests lie in the general area of computer systems.

Bibliography

[1] Ashley, Paul, Bradley Broom, and Mark Vandenwauver, "An implementation of a secure version of NFS including rbac," *Australian Computer Journal*, Vol. 31, Num. 2, 1999.

[2] Callaghan, Brent, Brian Pawlowski, and Peter Staubach, "NFS version 3 protocol specification," *RFC 1813*, Internet Engineering Task Force, 1995.

[3] Cattaneo, Giuseppe, Luigi Catuogno, Aniello Del Sorbo, and Pino Persiano, 'The design and implementation of a transparent cryptographic filesystem for UNIX," *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, USENIX, 2001.

[4] Cho, Young H. and William H. Mangione-Smith, "Specialized hardware for deep network packet filtering," *12th International Conference on Field-Programmable Logic and Applications*, ACM, 2002.

[5] Cho, Young H. and William H. Mangione-Smith, "Deep packet filtering with dedicated logic and read only memories," *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2004.

[6] Dharmapurikar, Sarang, Praveen Krishnamurthy, Todd Sproull, and John Lockwood, "Deep packet inspection using parallel bloom filters," *Proceedings of the 11th Symposium on High Performance Interconnects*. IEEE, 2003.

[7] Dubrawsky, I., *Firewall evolution – deep packet inspection*, http://online.securityfocus.com/infocus/1716, 2003.

[8] Goh, Eu-Jin, Hovav Shacham, Nagendra Modadugu, and Dan Boneh, "SiRiUS: Securing Remote Untrusted Storage," *Proceedings of the Tenth Network and Distributed System Security (NDSS) Symposium*, Internet Society (ISOC), pp. 131-145, February, 2003.

[9] McLaughlin III, Leo J., "Line printer daemon protocol," *RFC 1179*, Internet Engineering Task Force, 1990.

[10] IOzone, *IOzone filesystem benchmark*, http://www.iozone.org , 2005.

[11] Lowth, Chris, *TCP/IP connection cutter*, http://www.lowth.com/cutter , 2003.

[12] Mayo, Jean and Phil Kearns, "A secure untrusted advanced systems laboratory," *Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education*, 1999.

[13] Miltchev, Stefan, Vassilis Prevelakis, Sotiris Ioannidis, John Ioannidis, Angelos D. Keromytis, and Jonathan M. Smith, "Secure and flexible global file sharing," *Proceedings of the FREENIX Track: 2003 USENIX Annual Technical Conference*. USENIX, 2003.

[14] Neuman, B. Clifford and Theodore T'So, "Kerberos: An authentication service for computer networks," *IEEE Communications Magazine*, Vol. 32, Num. 9, pp. 33-38, 1994.

[15] O'Shanahan, Declan Patrick, *CryptoFS: Fast cryptographic secure NFS*, Master's thesis, University of Dublin, 2000.

[16] Powell, P. and J. Mason, "Lprng – An enhanced printer spooler system," *Proceedings of the Ninth USENIX Systems Administration Conference*, pp. 17-22, USENIX, 1995.

[17] Russell, R., *The netfilter project*, http://www.netfilter.org , 2005.

[18] Vlachos, K., N. Nikolaou, T. Orphanoudakis, S. Perissakis, D. Pnevmatikatos, G. Kornaros, J. A. Sanchez, and G. Konstantoulakis, "Processing and scheduling components in an innovative network processor architecture," *Proceedings of the 16th International Conference on VLSI Design*, IEEE, 2003.

Administering Access Control in Dynamic Coalitions

Rakesh Bobba[1] – NCSA, University of Illinois, Urbana-Champaign, IL
Serban Gavrila – VDG Inc., Chevy Chase, MD
Virgil Gligor – University of Maryland, College Park, MD
Himanshu Khurana – NCSA, University of Illinois, Urbana-Champaign, IL
Radostina Koleva – University of Maryland, College Park, MD

ABSTRACT

Dynamic coalitions enable autonomous domains to achieve common objectives by sharing resources based on negotiated resource-sharing agreements. A major requirement for administering dynamic coalitions is the availability of a comprehensive set of access control tools. In this paper we discuss the design, implementation, evaluation, and demonstration of such tools. In particular, we have developed tools for negotiating resource-sharing agreements, access policy specification, access review, wholesale and selective distribution and revocation of privileges, and policy decision and enforcement.

Introduction

In various collaborative environments such as alliances for research and development, health care, airline route management, public emergency response, and military joint task forces, autonomous domains form *coalitions* to achieve common objectives by sharing resources (e.g., objects and applications). Coalition resources may be privately or jointly administered and resource sharing is achieved by the distribution of permissions for coalition resources to coalition members based on negotiated resource-sharing agreements, or *common access states*. These coalitions are dynamic in that member domains may leave or new domains may join after coalition establishment. The focus of this work is on providing tools for administering access control in such coalitions.

Consider the following example of a coalition. Initially, three DoD/Intelligence domains form a coalition to respond to a threat situation relating to national security (e.g., monitoring transport of nuclear material, activities of terrorist groups). Examples of such domains can be Defense Intelligence Agency (DIA), National Security Agency (NSA), Defense Threat Reduction Agency (DTRA) and Central Intelligence Agency (CIA). These three domains decide to share privately owned resources (e.g., NSA shares a database of intercepted communication and the CIA shares intelligence reports) as well as jointly administered resources (e.g., integrated intelligence data and operation reports). Jointly administered resources are typically critical to coalition objectives and remain with the coalition even after the departure of member domains [16]. Access to jointly administered resources is one of the benefits individual domains derive from their membership in the coalition.

To obtain assistance from local authorities for a specific threat the DoD/Intelligence domains invite a civilian domain, say the Threat/Emergency Response Division of County C, to join the coalition. Upon resolution of the specific threat, County C leaves the coalition. Such domain join and leave events should not require the breakdown and setup of coalitions from scratch. Instead, access control tools must support dynamic domain joins and leaves – *Dynamic Coalitions* (DC) – in a seamless manner with ease and efficiency.

In this example, all four domains are collaborating to achieve a *common objective* (i.e., addressing threats to national security) that is desired by all domains and achievable by none of them individually. To achieve the common objective, each domain has a set of resources that are of interest to the others and is willing to share some or all of its resources with some or all of the other members.

It is important to note that the domains may not be willing to share these resources in the absence of this common objective; e.g., the DoD domains may be competitors for military tasks and would compete based on their available resources. Resource sharing by a domain with other domains means that the domain (administrator) grants access privileges for the resource being shared to the other domains. The sharing relationships among these domains are defined by the *access state* of the coalition. An access state is defined by the permissions each member domain has to the shared resources of that coalition.

Thus negotiating a *common access state* (CAS) means obtaining the agreement of each domain to share some of its data and applications with the other domains of the coalition. In addition, a CAS[2] of a coalition may include some jointly owned resources created during coalition operations, and a common policy for accessing these resources must also be negotiated.

[1]Author names listed in alphabetical order.

[2]Appendix B lists acronyms used in this paper.

The negotiation result is not merely a union of the contributed resources necessary to achieve a coalition objective. Instead, the negotiated CAS must satisfy both resource and permission constraints. In this work we consider all constraints that arise in the Role-Based Access Control (RBAC) model [8, 9, 10, 2]. For example, when the DoD domains initially form a coalition they may agree on a obligation constraint that requires at least one user from each domain to have administrative privileges for jointly administered resources.

Typically, these constraints arise from coalition objectives, access policies that are either jointly or privately enforced by autonomous domains, and resource-access requirements of coalition applications. The specification of negotiation constraints is an important part of any access-policy specification and drives the negotiation process; e.g., it determines the number of negotiation rounds and the convergence to and commitment of CAS. Hence, it must be defined in a precise manner [17].

A typical negotiation for this example would begin with the three DoD domains joining the coalition, specifying negotiation constraints, contributing resources that they are willing to share, proposing desired CAS, voting on the proposals based on satisfaction of constraints and extra-technological preferences, and committing to an agreed proposal. This negotiation process would be repeated when the civilian domain joins and leaves the coalition. Since the number of objects being negotiated and the number of rounds of negotiation could be large even in this small example, the negotiation process would be time-consuming and error-prone if undertaken manually. (In large coalitions such as those compromising tens of domains sharing hundreds of resources [22] this point is further emphasized.) Consequently, tools that fully (or at least partially) *automate the negotiation process* are needed. Automated negotiation tools would enable domain administrators to easily compose proposals for a CAS, verify that these proposals satisfy negotiation constraints, and commit them to production systems once agreed upon. Furthermore, for large-sized coalitions tools that enable administrators to *visualize the CAS* and to administer coalition resources via a graphical interface are very helpful.

In addition, *access review* tools are needed that would enable individual domains to review their local access state and verify that the state is secure prior to negotiation; i.e., that the state satisfies all local domain access policies. Based on this review, domains would be able to contribute resources to the coalition and compose proposals for CAS. Furthermore, access review for coalition resources may be undertaken at any point in time after coalition setup to (1) view the CAS along desired lines (e.g., subject permissions for a specific application, application access to specific objects), and (2) verify satisfaction of policies and constraints (e.g., Separation-of-Duty (SOD) policies, obligation constraints).

Once the CAS has been negotiated, tools that provide *wholesale, selective distribution and revocation* of access privileges are needed. Tools for wholesale distribution/revocation are needed as privileges must be granted to/revoked from *all* users of joining/leaving domains. Granting/revoking privileges to individual users would place undue overhead on administrators and, more importantly, would make it difficult to support coalitions where duration of domain membership could be smaller than coalition setup times using today's deployed technologies (e.g., weeks or months). Tools for selective distribution/revocation are needed as the tools must selectively target users of specific domains. One cannot, for example, exclude a member domain simply by modifying CA trust relations because those trust relations may be needed to share resources with that domain as part of another coalition.

There exists a need for coalitions in the commercial world as well as in government settings. What follows are two examples of commercial coalitions, taken from [15, 17]. First, consider a genetics research firm that discovers a gene sequence associated with a disease and establishes a coalition with a pharmaceutical company, two research hospitals and a Food and Drug Administration review board (FDA board) to find a cure using the gene sequence. Each domain shares some of its local resources with the coalition partners to achieve the coalition objective; e.g., the genetics firm contributes its gene sequence database, the pharmaceutical company provides a drug composition tool, the hospitals support clinical trials and give access to their patient databases (while preserving patient privacy by withholding sensitive patient information such as name, social security number, etc.), and the FDA review board shares a database of safety regulations. Given the impact of finding a cure, the coalition decides to jointly administer an application for remote consultation and drug analysis. This remote consultation and drug analysis application relies on a jointly administered secure group communication service. As coalition operations proceed, member domains may leave and new domains may join the coalition.

Another example of a coalition would be airline companies that collaborate to share various types of airlines routes in order to expand their market coverage. Sharing an individual route of a certain type implies that the airline domain that owns the route grants access permissions required to execute the route applications (e.g., reservations, billing, advertising) for that route to users of a foreign airline domain. Domains share routes (private resources) and may choose to jointly administer an auditing application that ensures adherence to coalition policies; e.g., policies on multi-hop route pricing and frequent flyer mile programs. As the coalition proceeds airline partners may leave or join the coalition.

In this paper we describe a suite of access control tools that we have developed for supporting dynamic coalitions. The tools employ the RBAC model and have been implemented in the Windows 2000 Server environment using Java. We have extended[3] an existing RBAC tool, called Multi-Domain Role Control Center (MDRCC), to provide access policy negotiation (i.e., CAS negotiation), specification, and review. The extended tool, MDRCC (Multi Domain Role Control Center), has been integrated with the Windows Active Directory for instantiating a negotiated CAS and to provide a Policy Decision Point (PDP) for resource providers. MDRCC provides a policy visualization language and a Graphical User Interface (GUI) that enables domain administrators to have a common view of coalition operations and to administer those operations directly from the interface. MDRCC has been integrated with an Attribute Certificate Authority (ACA) to provide wholesale, selective distribution and revocation of privileges. We use the Windows IIS web server as an example of a resource provider that enforces access policies for shared web pages. We have conducted experiments with our tools to show that they scale to support reasonable size dynamic coalitions and provide results of these experiments. We demonstrated the tools at Joint Warrior Interoperability Demonstration (JWID) 2004 and obtained valuable feedback.

The rest of this paper is organized as follows: We discuss related work, present the architecture of our system and discuss the salient features on our access control tools, describe the implementation of the tools, present experimental results, discuss lessons learned, and then conclude.

Related Work

Access control in distributed systems, in general, and in DCs, in particular has been extensively studied in the past. However, to the best of our knowledge ours is the first complete set of tools for access control in DCs that provide (1) support for negotiating access policies among domains for coalition set up, (2) wholesale selective distribution and revocation of privileges, (3) access review capabilities, and (4) support for jointly administering coalition critical resources.

Access Control in Dynamic Coalitions

The Secure Virtual Enclaves (SVE) [23] infrastructure allows multiple organizations to share resources while retaining organizational autonomy over local resources. SVE provides tools for (1) access policy specification via RBAC and (2) policy decision and enforcement via Access Calculators and Interceptor/Enforcers respectively. The SVE infrastructure is implemented in Java. Java RMI is used for intra-enclave communication and Ensemble group communication via JavaGroups Interface is used for inter-enclave communication. Resources in SVE are distributed applications based on Java RMI, Microsoft DCOM, Sun's Java Web Server and Microsoft's IIS.

The Yalta system [24] enables secure tuplespaces distributed across administrative domains. A tuplespace is a content-addressable shared memory. Entities in Yalta share information through a shared tuplespace that is distributed across administrative domains. Yalta provides for policy specification via Java's policy objects, policy decision and enforcement via Java Authentication and Authorization Service(JAAS) interface, and privilege distribution and revocation via a threshold CA and a Certificate Revocation Notification (CRN) service respectively. Tuplespaces in Yalta are implemented using JavaSpaces. The threshold CA is implemented in Java using the Yalta JCE provider, which, in turn, is implemented using Jini and Java RMI over SSL.

dRBAC [7] is a decentralized trust management and access control mechanism. The supporting infrastructure is implemented in Java and enables resource sharing across administrative domains. The tools provide for policy specification via RBAC, and provide privilege distribution and revocation and policy enforcement via a distributed network of wallets.

However, none of the above work (SVE, Yalta and dRBAC) provide tools for negotiating access policies, wholesale and selective distribution and revocation of privileges, reviewing access policy and joint administration of resources.

Phillips, et al. [21], describe a security model and enforcement framework that controls access to APIs of software that operate in a distributed environment running middleware like JINI or CORBA. The framework provides tools for policy specification via RBAC, policy decision and enforcement and privilege distribution and revocation via Unified Security Resource (USR). USR is a set of middleware resources (JINI and CORBA) that manage all mandatory access control (MAC) and RBAC meta-data for users, user roles and resources. However, the framework uses password based authentication that might cause longer coalition set up times. This work provides tools for access review but does not provide tools for negotiating access policies, jointly administering resources or wholesale selective distribution and revocation of privileges.

TrustBuilder [25] addresses the problem of credential negotiation for trust establishment between clients and resource servers. This work assumes extra-technological resource sharing agreement between domains and deals only with credential negotiation. This would not scale well in large coalitions where users have to negotiate credentials on every access. In our work we facilitate multi-party negotiations between domains to come up with resource sharing

[3]The initial version of MDRCC, called RCC, provided RBAC policy administration for an individual domain and was implemented in Visual Basic.

agreements and policies for member domains to gain access to those resources.

Access Control in Distributed Systems

Though the following work doesn't deal with coalitions explicitly, it is relevant to our work. Herzberg *et al.* [11] presents a Trust Policy Language (TPL) that allows organizations to define policies that map users to roles based on credentials presented by the user and those automatically retrieved by the system from credential repositories. RT [19] is a family of Role-Based Trust Management languages for representing policies and credentials in distributed authorization. KeyNote [4] is a decentralized trust management system that provides policy specification via PolicyMaker, policy decision via Keynote policy interpreter that takes the application's policy and credentials presented in the request and outputs a decision. It de-couples the application logic from access policy and authorization logic. The RBAC model employed by RCC may have fewer features when compared with TPL, RT, KeyNote, and dRBAC. However, RCC has an implementation that provides support for easy constraint specification and verification and an intuitive GUI for policy specification, visualization and administration.

Architecture and Access Control Tools

In Figure 1, we outline the architecture for administration of access control in a dynamic coalition comprising of two domains – which can easily be extended to *n* domains. We assume that each autonomous domain has its own Identity Certificate Authority (CA) that distributes identity certificates to users registered in that domain, and a container of users, resources (objects, applications), and privileges. To participate in the coalition each domain (administrator) installs the Coalition Resource Management (CRM) toolkit that consists of MDRCC, an Attribute CA, and a Secure Group Communication (SGC) toolkit. MDRCC provides specification of access control policies in the RBAC model, partially automated negotiation of the CAS, a visual common view of coalition operations, and access review capability. Once the CAS is instantiated on the container of users and resources, MDRCC provides a PDP via the container and administration of access policies for the users and resources via an intuitive GUI. The Attribute CA distributes (and revokes) attribute certificates (ACs) to both local and foreign domain users authorizing access to local domain resources. To jointly administer coalition resources, member domains establish a Joint Attribute Authority (JAA) that consists of an MDRCC module and a Joint Attribute CA (JACA), that is, an Attribute CA whose private key is shared among the member domains in a threshold manner [5]. As an example of a resource provider, coalition domains also establish local domain and coalition web servers that manage local and jointly administered applications (e.g., App O). The SGC component enables domains to communicate securely; e.g., to establish a CAS.

Policy Specification

MDRCC uses the RBAC access control model for policy specification. We chose the RBAC model for two reasons. First, it simplifies administration of access control for large systems by separating the

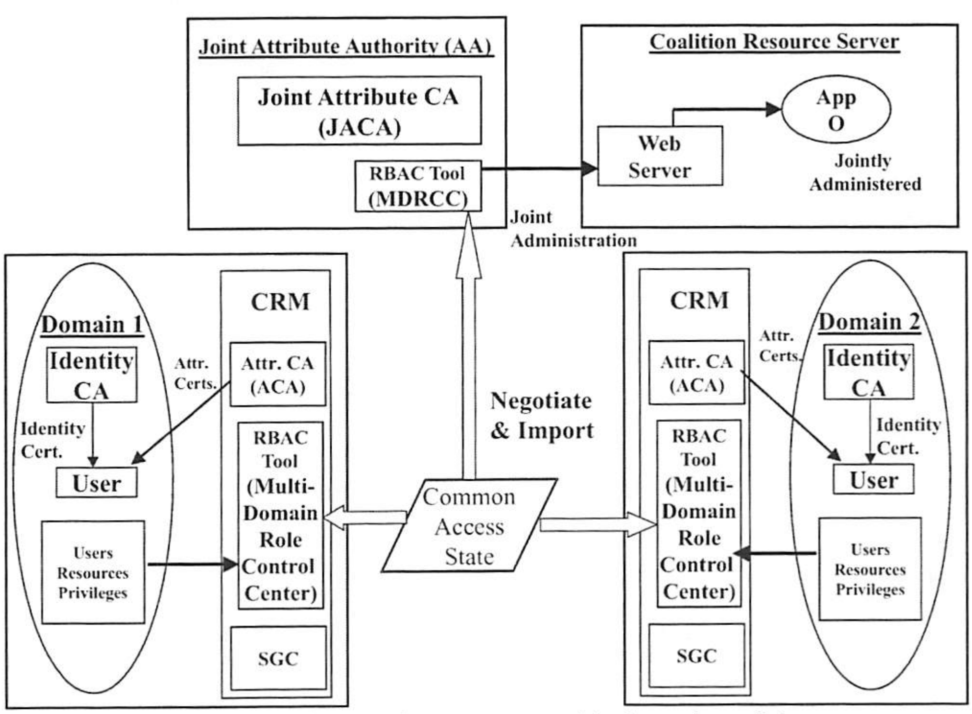


Figure 1: Architecture for access control in dynamic coalitions.

assignment of resource privileges to roles and users to roles [6]. Second, it supports easy specification and verification of constraints; e.g., Separation of Duty, and obligation constraints [8, 9, 2]. Though there are several policy specification tools based on RBAC (e.g., [7, 19, 11]) we chose MDRCC because it has been implemented, supports constraint specification and verification, and has a policy visualization language as well as an intuitive GUI that allows domain administrators to have a common view of coalition operations and to administer those operations from the GUI (e.g., add/remove users from roles).

Common Access State Negotiation

We use MDRCC and the SGC tool for partial automation of the process of negotiating a CAS among member domain. Administrators use MDRCC's GUI to encode negotiation constraints that can be either *global*, in which case they are known and agreed upon by all member domains, or *local*, in which case the constraints may remain private to some member domains. Types of constraints include Separation-of-Duty policies and cardinality constraints. Each domain administrator then uses MDRCC's intuitive GUI to specify access policies of resources they are willing to share and provides a view of these resources to other member domains. Any domain administrator then composes a proposal of a CAS (again using MDRCC's GUI) and sends the proposal to the other member domains. On receiving this proposal, domain administrators can view it and verify that it satisfies both global and local domain constraints (local constraints are also encoded using MDRCC's GUI). Domain administrators either register their vote on a given proposal or propose alternative CAS. Once all domains agree on a given proposal it is committed in that all domain administrators use MDRCC to instantiate the negotiated CAS on the local system; i.e., abstract objects and permissions in the CAS are mapped to actual objects and permissions of target systems.

Several approaches (e.g., agent-based) that fully automate multi-party access policy negotiations have been proposed (e.g., [3]) but have not yet been fully implemented. Though we are exploring such approaches, for the current system we chose the partially automated approach with MDRCC because, in practice, domain administrators would need to get involved in the negotiation as they would be responsible for administering the coalition and MDRCC provides these administrators with an intuitive GUI for the negotiation and administration.

Policy Decision and Enforcement

When a resource provider gets an access request it needs to verify the request against a specified access policy. To do so, the resource provider sends a request to a PDP and obtains the *policy decision*. In our solution this PDP is provided by the container where MDRCC instantiates the negotiated CAS. (As we will see later, Windows Active Directory is the container in our implementation that allows resource providers to send Lightweight Directory Access Protocol(LDAP) queries for policy decisions). This approach is scalable and is employed by many large-scale distributed systems; e.g., the XACML specification for web services [20]. After the resource provider obtains the policy decision it must enforce it. In our system we use a web server as an example resource provider and after obtaining a policy decision from the PDP the web server enforces it by allowing or denying access to the requested web page.

Distribution and Revocation of Privileges

Once a CAS is negotiated and instantiated, access privileges sanctioned in the CAS need to be distributed and we use attribute certificates to do so. We assign role memberships in attribute certificates and use a Public Key Infrastructure (PKI) for distribution and revocation of attribute certificates. Coalition users send requests to domain Attribute CAs for attribute certificates that grant them privileges to access the domain's resources. These requests includes the users' identity certificates. Attribute CAs verify the validity of the identity certificate and send requests to the PDP to verify the user's certificate request; i.e., to verify that the user has been assigned to the requested role in the CAS. Based on the reply from the PDP, Attribute CAs issue the attribute certificates. When users send access requests to resource providers with their attribute certificates, the resource providers verify the validity of the attribute certificates and query the PDP to verify if the roles assigned in the attribute certificates have the necessary privileges to access the requested resources. To revoke privileges, Attribute CAs simply revoke the attribute certificates and resource providers will no longer be able to validate the attribute certificates accompanying the access requests.

In addition to sharing privately owned resources, coalition members also benefit from jointly administering certain resources. Examples of such resources include intelligence reports, purchase orders for equipment, and financial data. In some coalitions, jointly administered resources may include auditing applications that are used to ensure that all domains are adhering to predefined access policies. These resources are typically critical for the coalition objectives and, therefore, must remain with the coalition even after the departure of member domains. To ensure the continuity of access to jointly administered resources in the presence of coalition dynamics, member domains setup a JAA. Since joint administration is consensus-based it is important to ensure that no single domain should be able to unilaterally define and modify access policies of a jointly administered resources [16]. To achieve this, the JAA comprises a Joint Attribute CA whose private key is distributed among the member domains; i.e., it uses threshold cryptography with each domain maintaining a share of the private key and using that

share for generating distributed signatures on attribute certificates [26]. The process of distributing and revoking privileges for jointly administered resources is similar to that for private resources except that signatures on the certificates are computed in a distributed manner with all domains participating to ensure consensus.

MDRCC is integrated with the domain Attribute CAs in each domain and with the Joint Attribute CA at the JAA to provide wholesale, selective distribution and revocation of attribute certificates. Administrators grant privileges to all users of a joining domain by just instantiating the CAS via MDRCC. MDRCC updates the container that acts as a PDP (w.r.t. certificate issuance) for the Attribute CA. Users of the joining domain can then request the Attribute CA (of the domain administering the shared resource) for attribute certificates before their first access to the resource and use the issued attribute certificate to access the resource there after. Whenever a user is issued an attribute certificate the Attribute CA registers it with its local MDRCC. Domain administrators can revoke all privileges of all users of a leaving domain just as easily as revoking the privileges of an individual user – with just a few clicks. When an administrator revokes user privileges via MDRCC's GUI, a request is sent to the local Attribute CA to revoke the respective attribute certificates. The Attribute CA revokes the attribute certificates and publishes a CRL.

Access Review

MDRCC uses the capabilities of the underlying RBAC language and a user-friendly GUI for access review and visualization. Domain administrators can (1) perform an access review along the desired lines (e.g., per-subject review, per-object review), and (2) verify the satisfaction of policies and constraints (e.g., Separation-of-Duty policies, obligation constraints) using MDRCC's interface. Also, once a policy is specified in MDRCC using the interface, MDRCC continuously monitors all future administrative actions for compliance.

Component Design and Implementation

Our tools are primarily implemented in Java and have been tested in Windows 2000 server environment. In this section we describe the implementation of the main components of our tools and the interaction between those components. In order for all the above components to inter-operate in a coalition environment, inter-domain trust relations must be established. For example, for MDRCC to verify signatures on CAS before instantiating it, MDRCC needs to have public-keys/certificates of the administrators and also trust them. These trust relations are set up manually before coalition set up using Windows Certificate Trust Lists that are easily composed and shared among the various components.

Multi-Domain Role Control Center (MDRCC)

MDRCC is a Java implementation of a Role-Based Access Control (RBAC) model [1] extended with general role hierarchies across multiple domains, static separation of duty and cardinality constraints, and advanced access review facilities. In MDRCC, each user or role is owned by a particular domain and the roles owned by a domain form a hierarchy based on the specified role inheritance relation. In each domain, the base of the role hierarchy is a special role called the domain base role. Users of any domain can be assigned to a role in a given domain. Roles are assigned abstract permissions to abstract objects. In addition, MDRCC includes data structures for (1) mapping selected portions of a role hierarchy within a domain to user accounts and groups on the domain controller, and (2) mapping the authorization information at the enterprise level (that is in terms of abstract objects and abstract permissions) to actual objects and permissions on the resources resident in various heterogeneous systems in the format required by native access control structures.

MDRCC is a three-tiered component that is made up of the following tiers: a Presentation Layer, an Application Logic Layer, and a Data Layer. The presentation layer comprises the MDRCC client(s), the application layer comprises the MDRCC server and MDRCC agents (resident in various target systems), and the data layer comprises the data repository. An MDRCC client provides a GUI for displaying the multi-domain RBAC model graph (or role graph), capturing user (Administrator) actions and sending them to the (remote) MDRCC server. The MDRCC server receives user (administrative) commands from the client, and executes them by accordingly updating the data layer. The MDRCC server is also responsible for mapping selected sub-graphs of the role graph (called views) to user accounts and groups on heterogeneous hosts (called also target systems), and for mapping abstract objects and role permissions to actual objects and permissions structures (e.g., ACLs) on those hosts. For these tasks, MDRCC uses agent software running on each host to create/delete groups and user accounts, and set up ACLs, according to commands received from the MDRCC server. The data layer consists of a directory service, which stores, retrieves, and protects the actual multi-domain RBAC data; i.e., the domain information, the user and role sets, the various relations, the abstract objects, and the mappings between multi-domain RBAC data and target system data. In our system, the Windows 2000 Active Directory provides the directory service; i.e., it is the data layer. The communication between MDRCC client and server is via SSL and is implemented using JSSE packages.

An MDRCC client not only provides visual access policy representation (see Figure 2) by displaying the role graph but also provides a very intuitive and user

friendly GUI for administrative actions; e.g., creating roles, assigning permission to roles. For negotiating a CAS, domain administrators use an MDRCC client to build a role graph (a CAS proposal) from the known set of resources that domains are willing to share and export the role graph into a compact, machine readable access state file. Other domain administrators then import the proposed CAS file and visualize it. They can verify that the proposed CAS satisfies local permission-based constraints using MDRCC's constraint verification facilities, and then either vote upon it or propose a new CAS. MDRCC currently supports the specification and verification of Static Separation of Duty (SSD) and cardinality constraints [8].

Once a CAS has been agreed upon all the administrators sign it with their private keys; i.e., to enforce agreement. The signed CAS is then imported into MDRCC at each domain and also at JAA. The CAS is then instantiated; i.e., abstract objects and permissions in the CAS are mapped to actual objects and permissions of target systems after verifying the signatures on the CAS.

Figure 2 shows a typical view of MDRCC's GUI and the Attribute CA GUI. MDRCC represents users and roles of the CAS as ovals of different colors, each color corresponding to a different domain. Roles are represented with solid ovals and users with solid ovals enclosed in an outer oval. Here we see two users from county C and a user from domain1 assigned to role "Manager". By right-clicking on any oval a drop-down menu is displayed which allows different operations for manipulating the MDRCC graph, setting and viewing role and user permissions, revoking user and viewing Separation-of-Duty constraints. Further options are available from the menu bar at the top which allow for changing the domain view, so that any domain can have a view of the entire CAS.

For access review administrators can perform either a per-subject review (reviewing all the permissions a subject has to all objects) or a per-object review (reviewing all users who have access permissions to a given object) with just a few clicks. For example, by right clicking on a user node in the graph and clicking "view permissions" in the drop down menu performs a per-subject review. Figure 3 shows how MDRCC allows for easy per-subject review of privileges. By left-clicking on the oval of a particular user, MDRCC displays all of the user's memberships to roles across all domains of the coalition. By right-clicking on each role and selecting view permission we can see the resources the user gets access to through that particular role and also the operations the user is allowed to perform on the resource. In some cases there is a need to view all of the user's accesses to coalition resources directly without viewing his role memberships, and MDRCC allows for viewing those accesses by right-clicking on the user name and selecting view permissions.

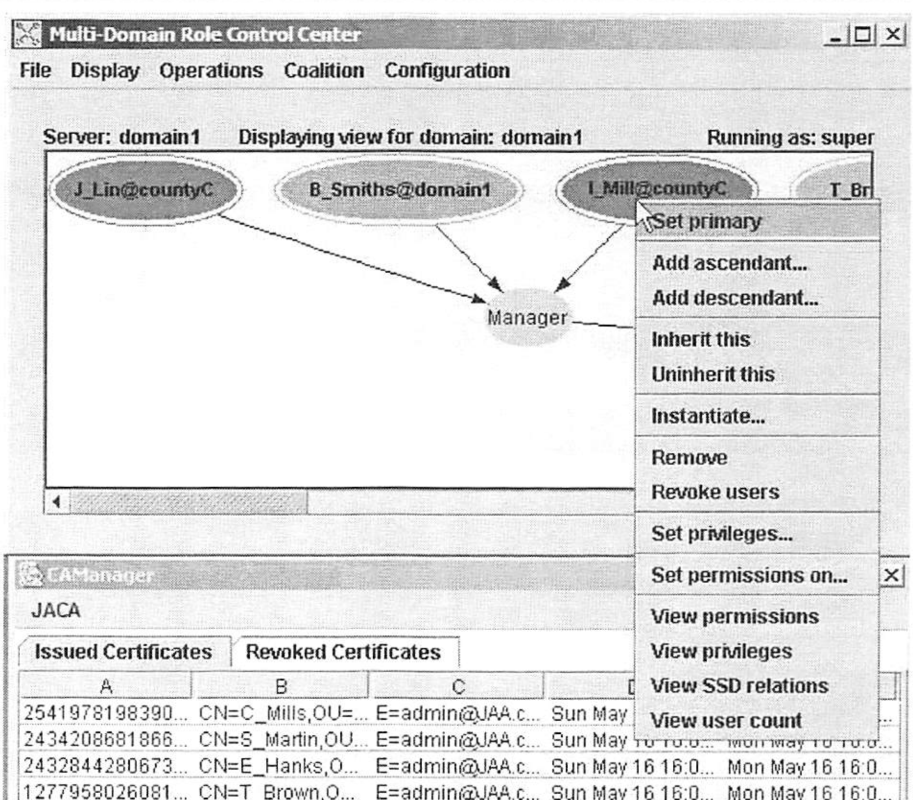


Figure 2: General view of MDRCC and Attribute CA during coalition operation.

Figure 4 shows MDRCC's interface for viewing and defining static Separation-of-Duty (SSD) permissions. The upper left quarter shows a list of defined constraints. A constraint is defined by a *role set* and a *threshold*. In this case we have one constraint with a role set called *ssd1* and a threshold of *two*. When a constraint is selected from the list, the roles that the role set of this constraint encompass are shown in the upper right quarter. The meaning of the constraint, *ssd1:2*, shown in the figure is the following: no more than two roles of the role set *ssd1* can have a common user. In Figure 3 we can see that this constraint is violated as user *J_Doe@domain3* belongs to more than two roles of the role set *ssd1* and MDRCC shows an appropriate error message and does not proceed further until the conflict is resolved. On the lower right quarter of Figure 4 are the coalition roles, for selection when creating new constraints.

Certification Authorities

Our system comprises three different kinds of CAs namely, domain Attribute CA, Joint Attribute CA and domain Identity CA. We assume that every coalition member has an Identity CA prior to coalition formation and hence do not deal with it here. However, for our testing and experiments we configured the CA service in windows 2000 server environment to act as an identity CA for member domains.

Attribute CA. Each coalition member domain has an Attribute CA that issues attribute certificates to both local domain and foreign domain users authorizing access to local domain resources. Attribute CA is a three-tiered component comprising the following tiers: a Presentation Layer, an Application Logic Layer, and a Data Layer. The presentation layer comprises the Attribute CA console (GUI), the application logic layer comprises the CA logic, and the data layer consists of key stores, issued certificates, published Certificate Revocation Lists (CRLs) and persistent data structures. The Attribute CA console provides an interface for administrative actions (e.g., starting the CA, stopping the CA, generating new keys, viewing currently issued certificates, viewing revoked certificates) and is implemented using *javax.Swing package*. The Application Logic Layer deals with issuing certificates, revoking certificates, and generating keys and is implemented using the Bouncy Castle Crypto API [13] and Java JCE packages.

Joint Attribute CA. Joint Attribute CA resides at JAA and issues attribute certificates to coalition users authorizing them access to jointly administered resources. Similar to Attribute CA, Joint Attribute CA is a three-tiered tool that comprises a Presentation Layer, an Application Logic Layer, and a Data Layer. But unlike Attribute CA, which uses a standard RSA cryptosystem, Joint Attribute CA uses a shared-RSA cryptosystem [5] and hence its Application Logic Layer and Data Layer are distributed among the coalition member domains. (In shared public-key cryptosystems the public key is owned by multiple principals with each principal having a share of the corresponding private key. The shared private key is generated in a distributed manner by all the participating domains (principals) and each domain retains a share

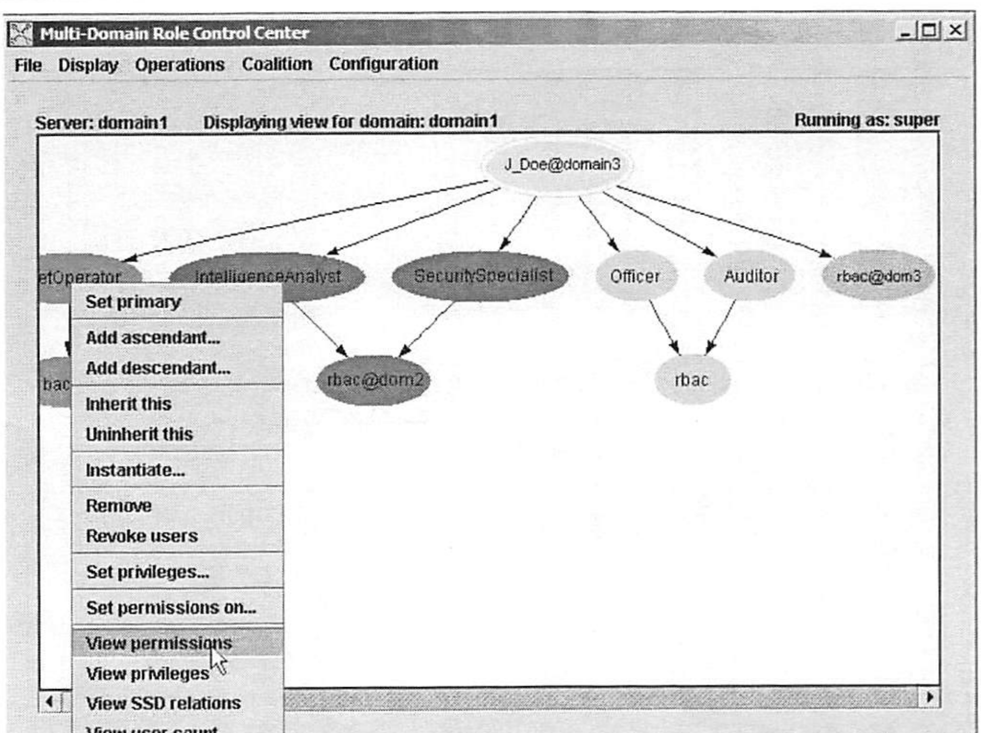


Figure 3: Per-subject review capability of MDRCC.

of the private key). These layers consist of (1) signature servers that reside in each domain for maintaining shares of the private key and signing certificates and CRLs and (2) a coordinator that resides at the JAA for coordinating the operations of the signature servers; i.e., the signature operation is distributed and is performed by the signature servers but is composed into a valid RSA signature by the Joint Attribute CA coordinator (see Appendix A for details on the employed shared-cryptosystem). The application logic layer is implemented using the Bouncy Castle Crypto API [13] and the Yalta JCE provider [24], which implements Boneh and Franklin's shared-RSA cryptosystem [26] with signature servers that communicate over Java RMI. We augmented the signature servers with a policy module (i.e., to enable policy checking capabilities). The Attribute CA/Joint Attribute CA GUI in Figure 2 displays the issued and revoked attribute certificates issued to both local users (in case of Attribute CA) and users from other domains. It also has menu options that allow the operation of Attribute CA to be suspended and started as needed.

Certificate issuance for Attribute CA and Joint Attribute CA is provided via a web server. Users authenticate to the web server using their domain identity certificates and upload their certificate request files, which are PEM encoded PKCS 10 files. Users then download issued PEM encoded X.509 V3 certificates. Both Attribute CA and Joint Attribute CA check with local MDRCC (which acts as a PDP for certificate issuance) and register issued and revoked certificates with MDRCC. In Joint Attribute CA the signature servers are also capable of checking certificate requests with local MDRCC to ensure that they comply with the CAS. Communication between the CAs (Attribute CA and Joint Attribute CA, both coordinator and signature servers) and MDRCC is via LDAP. When an administrator instantiates a negotiated CAS on the Active Directory, a PDP is automatically created that allows both Attribute CA and Joint Attribute CA to verify certificate requests providing wholesale distribution of privileges. If the instantiated CAS includes the departure of a domain then the administrator can revoke all users of that domain using the MDRCC client's GUI. This action results in the MDRCC Server sending a signed list of serial numbers (of certificates) that need to be revoked to the Attribute CA/Joint Attribute CA. Attribute CA/Joint Attribute CA process the request and publish a CRL on the web server, the URL for which is included in all certificates issued by the CAs. Attribute CA/Joint Attribute CA then register the revoked certificates with MDRCC via LDAP. Thus MDRCC and Attribute CA/Joint Attribute CA provide wholesale, selective distribution and revocation of privileges.

Figure 5 shows MDRCC's interface that allows for selective privilege revocation. Any domain (administrator) can selectively revoke the users from another domain (or all other) in a single operation. This figure in particular shows the MDRCC's ability for selective revocation where the privileges of users from another domain are revoked only for a specific role and not all roles in general.

Resource Server

In our implementation we use web pages as example resources and use the Windows IIS 5.0 web server as the resource server. To access web pages users submit attribute certificates to the web server, which checks the validity of the certificates by verifying the signature, checking the expiry time, and checking against the CRL for revocation information. After verifying the certificate (using the certificate verification functions built-in with IIS 5.0), the web server

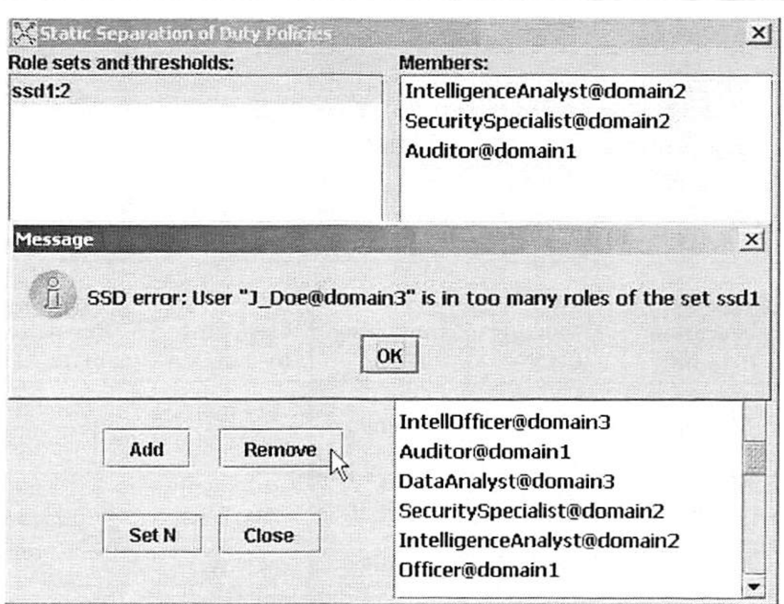


Figure 4: RCC's interface for specifying Separation-of-Duty constraints.

queries MDRCC (via LDAP) and grants or denies the request based on the authorization reply from MDRCC.

Secure Group Communication (SGC)

Secure group communication is needed in dynamic coalitions to aid administrators in negotiation during coalition set up and for smooth operation of the coalition after it is set up. Since the SGC requirement is not unique to dynamic coalitions we chose to use existing SGC tools. We use both Secure Spread [14] and Secure E-mail List Service [18]. The former provides secure, reliable multicast while the latter provides ease of use of e-mail in a secure manner.

Experimental Setup and Results

We set up experiments to measure the time taken for setting up a coalition of three domains and for domain join and leave events using our access control tools. Each domain in the experiment has ten applications that it shares with the coalition (private resources), ten roles that have permissions for the shared applications, and fifty users that get assigned to foreign domain roles and jointly administered roles for access to shared applications. Coalition members set up a JAA with ten jointly administered applications and ten roles with permissions for these applications. Coalition setup time is shown in Table 1 and includes time for importing a negotiated CAS (we do not include the time for negotiating a CAS as it is largely influenced by extra-technological decisions and actions), shared-key generation (since distributed shared-key generation is probabilistic the key generation times shown in the table are averaged generation times averaged over 10-15 runs), and certificate generation and distribution.

For coalition dynamics, we measured the time taken for a domain to join an existing coalition of three domains and for a domain to leave a coalition of four domains. The results are shown in Table 1. In case of domain departure we assume that the departing domain relinquishes its share of private key and hence there is no need to generate a new key. Note that this assumption only holds as long as the number of member domains that have left the coalition after the last key generation is not greater than *floor*(n/2) where n is

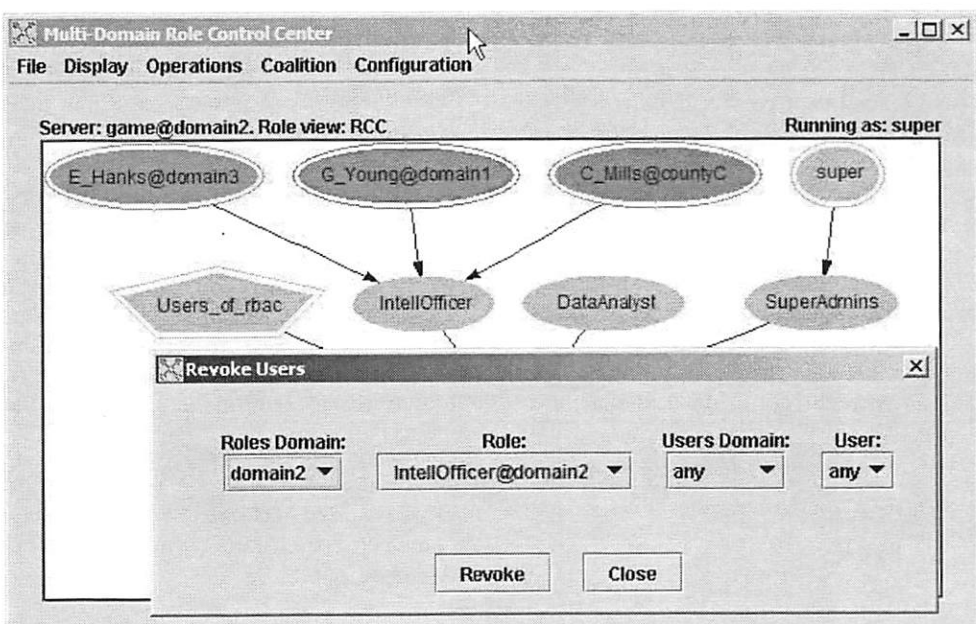


Figure 5: MDRCC's interface for wholesale, selective revocation.

	Shared-key generation (1024 bits)	Exporting CAS	Importing CAS	Cert. Dist. by JACA	Cert. Dist./ Revok. by domain ACAs	Total time
Coalition set-up with 3 domains	41 min	N/A	11 min	14 min (500 certs)	3 * 6 min (3 * 500 certs)	84 min
Domain Join	46 min	10 sec	16 min	17.5 min 600 certs	4 * 2 min (4 * 125 certs)	88 min
Domain Leave	N/A	10 sec	11 min	N/A	3 * 2 min (3 * 125 certs)	17 min

Table 1: Experiments for Coalition Setup, Domain Join and Domain Leave.

number of domains in the coalition at the time of key generation. Otherwise the member domains that left the coalition can collude to compromise the private key [5]. A domain leave event with key generation takes about the same amount of time as a domain join event. As can been seen from the measured times coalition set up and dynamics take only few hours at most for moderate sized coalitions there by illustrating the scalability of our access control tools.

Lessons Learned

The tools were demonstrated at various DARPA Principal Investigator (PI) meetings and most recently at the Joint Warrior Interoperability Demonstrations (JWID) in June 2004 [12]. JWID is an annual Chairman of the Joint Chiefs of Staff event that enables U. S. Combatant Commands and the international community to investigate command and control, communications and computer (C4) solutions that focus on selected core objectives. JWID 2004 assessed capabilities and technologies that allow information sharing with Homeland Security and Homeland Defense partners. This particular demonstration experience gave us a lot of useful feedback on our tools. JWID 2004 was conducted in a simulated operational environment. Our goal was to train users to use our tools who, in turn, demonstrated the usability of the tools to military personnel.

Our training experiences can be summarized in the following four lessons. First, we could train users (who had little security knowledge) in a very short amount of time to use our tools because 1) the tools enable visual representation of the CAS and provide an intuitive GUI and 2) we had generated and provided the users with extensive documentation in the form of task guides that take the users through execution of common coalition administrative operations step by step with the aid of images and animations. Second, the lack of security knowledge combined with multiple windows/interfaces of our tools intimidated the users. The users were expecting one unified tool (interface) but in reality we have a set of tools that integrate functionally and have individual interfaces. Third, in spite of our best efforts to test the system prior to the demonstration we still over looked flaws that were (inadvertently) detected in user training. For example, we did not prevent against multiple instantiations of MDRCC and users started multiple instances of MDRCC leaving the data layer in unknown states. Fourth, though our set of tools demonstrated the functional capabilities expected of them by the evaluators at JWID 2004, they were not as effective as we would have liked largely because users of the tools need to be knowledgeable of the underlying technologies in order to use them effectively. Our efforts in making them user friendly and providing good documentation helped us to a certain extent but did not eliminate the necessity to educate the users about the underlying technologies.

Conclusions

We have developed tools for administering access control in dynamic coalitions. In particular, tools for negotiating resource-sharing agreements, access policy specification, access review, wholesale and selective distribution and revocation of privileges, and policy decision and enforcement. Our experiments demonstrate that these tools scale to handle reasonable size coalitions. Demonstrations of the tools at JWID 2004 provided an opportunity for user training and obtaining useful feedback. Though we demonstrated our tools in Windows environment they can easily be ported to other platforms. In the future, we plan to implement the tools over web services for platform independence and rapid prototyping of enhanced features and capabilities.

Acknowledgements

We would like to thank Adam Moskowitz and the anonymous reviewers for their helpful comments and suggestions. Part of the first and fourth authors' work was funded by the Office of Naval Research under contract N00014-04-1-0562. The second, third and fifth authors' work and part of the first and fourth authors' work was funded by the Defense Advanced Research Projects Agency and managed by the U. S. Air Force Research Laboratory under contract F30602-00-2-0510. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research, Defense Advanced Research Projects Agency, U. S. Air Force research Laboratory or the United States Government.

Bibliography

[1] ANSI INCITS 359-2004, *Information technology: Role based access control*, 2004.

[2] Ahn, Gail-Joon and Ravi Sandhu, "Role-based authorization constraints specification," *ACM Transactions of Information System Security*, Vol. 3, Num. 4, pp. 207-226, 2000.

[3] Bharadwaj, Vijay G., and John S. Baras, "Towards automated negotiation of access control policies," *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, p. 111, IEEE Computer Society, Washington, DC, 2003.

[4] Blaze, Matt, Joan Feigenbaum, and Angelos D. Keromytis, "Keynote: Trust management for public-key infrastructures (position paper)," *Proceedings of the 6th International Workshop on Security Protocols*, pp. 59-63, Springer-Verlag, London, 1999.

[5] Boneh, Dan, and Matthew Franklin, "Efficient generation of shared RSA keys," *Lecture Notes in Computer Science*, Vol. 1294, 1997.

[6] Ferraiolo, David, Janet Cugini, and Richard Kuhn, "Role-based access control (rbac): Features and motivations," *Proceedings of 11th Annual Computer Security Application Conference*, pp. 241-248, Springer-Verlag, New Orleans, LA, 1995.

[7] Freudenthal, Eric, Tracy Pesin, Lawrence Port, Edward Keenan, and Vijay Karamcheti, "drbac: Distributed role-based access control for dynamic coalition environments," *Proceedings of International Conference on Distributed Computing Systems*, pp. 411-420, Vienna, Austria, 2002.

[8] Gligor, Virgil, Serban Gavrila, and David Ferraiolo, "On the formal definition of separation-of-duty policies and their composition," *RSP: 19th IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, 1998.

[9] Gligor, Virgil D. and Serban I. Gavrila, "Application-oriented security policies and their composition (position paper)," *Proceedings of the 6th International Workshop on Security Protocols*, pp. 67-74, Springer-Verlag, London, UK, 1999.

[10] Gligor, Virgil D., Himanshu Khurana, Radostina K. Koleva, Vijay G. Bharadwaj, and John S. Baras, "On the negotiation of access control policies," *Revised Papers from the 9th International Workshop on Security Protocols*, pp. 188-201, Springer-Verlag, London, UK, 2002.

[11] Herzberg, Amir, Yosi Mass, Joris Michaeli, Yiftach Ravid, and Dalit Naor, "Access control meets public key infrastructure, or: Assigning roles to strangers," *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, p. 2, IEEE Computer Society, Washington, DC, 2000.

[12] https://www.cwid.js.mil/public/cwid05fr/START_HERE.html .

[13] http://www.bouncycastle.org/ .

[14] http://www.cnds.jhu.edu/research/group/secure_spread/ .

[15] Khurana, Himanshu, Serban Gavrila, Rakesh Bobba, Radostina Koleva, Anuja Sonalker, Emilian Dinu, Virgil Gligor, and John Baras, "Integrated security services for dynamic coalitions," An extended exposition abstract in *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX III)*, 2003.

[16] Khurana, Himanshu, Virgil Gligor, and John Linn, "Reasoning about joint administration of access policies for coalition resources," *ICDCS '02: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, p. 429, IEEE Computer Society, Washington, DC, 2002.

[17] Khurana, Himanshu and Virgil D. Gligor, "A model for access negotiations in dynamic coalitions," *WETICE '04: Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'04)*, pp. 205-210, IEEE Computer Society, Washington, DC, 2004.

[18] Khurana, Himanshu, Adam Slagell, and Rafael Bonilla, "Sels: a secure e-mail list service," *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, p. 306-313, ACM Press, New York, NY, 2005.

[19] Li, Ninghui, John C. Mitchell, and William H. Winsborough, "Design of a role-based trust-management framework," *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, p. 114, IEEE Computer Society, Washington, DC, 2002.

[20] Moses, Tim, "Core specification: extensible access control markup language (xacml) version 2.0," *OASIS Specification Document*, http://docs.oasis-open.org/xacml/access_control-xacml-2_0-core-spec-cd-04.pdf, 2004.

[21] Phillips, Charles E., Steven A. Demurjian, and T. C. Ting, "Information sharing and security in dynamic coalitions," *Proceedings of IEEE Information Assurance Workshop*, 2002.

[22] Phillips, Charles E., T. C. Ting, and Steven A. Demurjian, "Information sharing and security in dynamic coalitions," *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, p. 87-96, ACM Press, New York, NY, 2002.

[23] Shands, Deborah, Richard Yee, Jay Jacobs, and E. John Sebes, "Secure virtual enclaves: Supporting coalition use of distributed application technologies," *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2000.

[24] Smith, T. J., G. T. Byrd, W. Xiaoyong, X. Hongjie, K. Thangavelu, R. Wang, and A. Shah, "Yalta: A dynamic pki and secure tuplespaces for distributed coalitions," *Proceedings of 3rd DARPA Information Survivability Conference and Exposition*, Washington, DC, 2003.

[25] Winslett, Marianne, Ting Yu, Kent E. Seamons, Adam Hess, Jared Jacobson, Ryan Jarvis, Bryan Smith, and Lina Yu, "Negotiating trust on the web," *IEEE Internet Computing*, Vol. 6, Num. 6, pp. 30-37, 2002.

[26] Wu, Thomas, Michael Malkin, and Dan Boneh, "Building intrusion tolerant applications," *Proceedings of the 8th USENIX Security Symposium*, pp. 79-91, Washington, DC, 1999.

Appendix A: Shared RSA Public-Key Cryptosystem

In this section we discuss the use of shared public key techniques for generation and distribution of attribute certificates granting access to jointly owned resources.

Shared RSA Public Key Generation Algorithm

Here we review some of the features of the shared RSA public-key generation algorithm of [5].The algorithm enables n domains to generate a modulus $N = pq$ and exponents e and d. At the end of the computation all domains are convinced that N is the product of two primes, however none of them know the factorization of N. The public exponent e is made public while d is shared among the domains in a way that enables m-out-of-n threshold signature generation. That is, m domains are able to issue a certificate without reconstructing the key d. This also implies that an attacker who penetrates at most m-1 domains is unable to obtain any information about the private key. From the point of view of collusion the algorithm is $(n-1)/2$ private. That is, even if $(n-1)/2$ domains share the information they learn during the protocol, they will still not be able to recover the factorization of N or the private key d.

Joint Signatures with Distributed Private Key Shares

For joint administration of access policies, the public key K_{AA} of Joint Attribute CA (see Figure 1) is generated using the shared key generation algorithm resulting in private key shares that are distributed among all member domains (i.e., a n-of-n threshold sharing of the private key K_{AA}^{-1}). Once the public-key K_{AA} has been generated, all domains must apply a joint signature algorithm with their private key shares in order to sign attribute certificates with the private key K_{AA}^{-1}. The joint signature algorithm involves the Joint Attribute CA coordinator sending a message (the attribute certificate) to all the signature servers (co-signers) with the message M to be signed and a key ID comprising the hash of N and the public exponent e. Each of the co-signers then apply their corresponding private key shares d_i to compute $S_i = M^{d_i} \bmod N$ and send the computations back to the coordinator. The coordinator then computes the message signature $S = \prod_{i=1}^{n} S_i \bmod N$. This joint signature protocol is illustrated in [26]. Using this joint signature algorithm, the domains sign threshold attribute certificates (and CRLs) distributed by Joint Attribute CA. The joint signature algorithm also works for a threshold sharing of the private key; i.e., the private key K_{AA}^{-1} is shared in a t-of-n manner among the member domains. The algorithm is similar to the case above except that when the coordinator only needs computations from t servers and can compute the message signature $S = \prod_{i=1}^{t} S_i \bmod N$.

Appendix B: List of Acronyms

This appendix contains a list of acronyms used in the paper.

ACA Attribute Certificate Authority
ACL(s) Access Control List(s)
CA(s) Certificate Authority(ies)
CAS Common Access State
CRL(s) Certification Revocation List(s)
GUI Graphical User Interface
IIS Internet Information Service
JAA Joint Attribute Authority
JACA Joint Attribute Certificate Authority
LDAP Lightweight Directory Access Protocol
MDRCC Multi-Domain Role Control Center
PDP Policy Decision Point
PKI Public Key Infrastructure
RBAC Role-Based Access Control
SGC Secure Group Communication

Manage People, Not Userids

Jon Finke – Rensselaer Polytechnic Institute

ABSTRACT

Despite the title, this is not about managing people, but rather managing the enterprise data about the people, especially in defining the relationship between a person and the organisation and controlling functions based on that relationship, or what some people might refer to as identity management.

Single sign-on is an attractive goal for many organisations. When you include parking gates and badge readers on building entrances, the problem gets even more interesting. As we expand our deployment of wireless access points and publically accessible network jacks, the need to require authentication for access to our virtual world grows stronger. With the need for authentication, so grows the demands on the systems that provide authentication and authorisation, especially in the area of managing who gets access and revoking that access at the appropriate time. Concurrently, with the rising interest in physical security of our facilities, the need for authentication and controlling access to our physical world is also growing. This also requires tools and systems to manage the people and their status and privileges.

Both of these issues share many common attributes and can be well addressed by merging them into a single system to manage people information, and from that, access to the virtual (network) world as well as the physical world. By combining these projects, we are able to take advantage of the mandate (and administrative support) to identify all of the people on our campus to provide physical access control, and so, manage our virtual world. We will also attempt to define a somewhat generic or standard methodology for doing this with our particular business rules and requirements confined to a few limited and specific areas.

While the technical issues are challenging, the more daunting task comes with negotiating the institutional politics and getting adequate "buy in" from the appropriate departments to provide the people and resources willing to operate and use the eventual technical solutions. This paper discusses both the social and technical aspects of those solutions.

Introduction

There have been a number of systems developed over the years to manage user accounts, *Moira* [12] in 1988 (part of MIT's Project Athena), to *Accountworks* [2] 10 years later and many others. In many ways, this process is pretty well understood, and many mechanisms exist for taking a user account and getting it the end system, be it Windows 2000 [7] and others [10, 11, 3]. In *A Retrospective on Twelve Years of LISA Proceedings* [1], the authors identified 23 papers dealing with account management. Under the section *Future Research Opportunities* they write:

> Surveying account creation practices would help identify why no tool has evolved as superior despite many papers on this subject. We believe this is because of unrecognised differences in the requirements at each site. With all of the requirements explicitly described, it should be possible to build a universal tool.

Like the organisations cited above, we here at RPI, have been developing our own system (Simon) for automatic creation and management of computer accounts. Since Simon's inception in the early 1990s, it has evolved to maintain our telephone directory [5] and ID card and parking control systems, as well as providing a consistent data feed [9] of people and directory information to a number of other systems on campus. This paper is in part a case study on how we addressed both the political and technical issues.

The basic objective of all of these systems is to automatically create and maintain computer accounts for all students, employees, contractors, etc. as needed. Given a good data feed, this is basically a technical exercise, which has been documented in the references cited above and many other places. But in all of these cases, the differences come from that prerequisite; the good data feed (or feeds), ideally from the system of record for that data element. While every site is different, I feel that they all share some commonality and I hope to outline some approaches that may be applicable to your specific site. One of the challenges we face, is that for the most part, no one outside of the IT department really cares all that much about solving this problem. After all, it is IT's problem to solve, and it doesn't really impact people outside of the IT department at all. We found that by expanding the scope of our project from just computer accounts, to other "people" related activity such as the phone directory or the ID card system, we were able to get

the interest and assistance of some of the key players outside of the IT department.

From a technical standpoint, while I don't propose to describe the universal tool in this paper, I do hope to describe a close to universal methodology that can be applied to most educational or corporate[1] environments. As suggested in the above quote, the problem is to define a generic enough tool that it can handle the requirements of different installations, while still allowing the easy application of the appropriate business rules with minimal changes to the code base.

For the most part, dealing with data feeds from Human Resources (for employee data) and the Registrars office (for student data) is well understood. The challenge we face is dealing with the "none of the above" categories.[2] In an ideal world, we would find a way to push that maintenance load out on the departments and offices. From the IT standpoint, we don't really have the juice (we can't mandate cooperation) to get departments to maintain status information, but by merging the computer account management with the more general ID card and directory processes, we are able to get sufficient interest to maintain the status information.

In this paper I hope to first describe how we addressed the political and social issues for implementing this system. I will discuss some of the arrangements that were made, and some of the arguments that were used to convince people in other departments and divisions that they needed to be part of this project and provide the operational support to make it all work. In addition, I will describe some of the philosophy and guidelines we adopted dealing with sources and maintenance of information, and some of the pitfalls we encountered.

Moving from the human realm to the technical realm, I will next describe a general schema for managing "people" (and "department") information, and given that, then describe a general, adaptable system for managing and delegating all of the oddball categories of people, and provide code examples that do NOT have our business rules built in, but are easily adaptable for any site. From this, we derive a general mechanism to evaluate and integrate the "status" of a person, in an easily adaptable and flexible manner. This in turn drives the business needs of the enterprise, be it computer accounts, ID cards, directory entries, or whatever the access and authentication needs are.

Obtaining Buy In from Others

The fall of 2003 brought to the fore, the issue that would finally get serious administrative support behind the unified people status project, and possibly a hot button topic just about anywhere, Parking. In our case, it was the installation of parking gates. Now it really mattered if someone's ID card expired – they would be denied access to our campus. In point of fact, there were at least 40 different departments or units identified as "stake-holders"[3] in this project. The parking lot access project let us bring some focus to the project and get a reasonable number of key people into meetings.

Authoritative Data Sources

As part of an earlier data feed project, we found it useful to identify the authoritative data source or "system of record" for each type of data. Despite having both student and employee information in our central administrative database (SCT Banner),[4] it quickly became obvious that although most people were there, not all people could be found there. Thus after much discussion, it was decided that although Banner would remain the system of record for student and employee data, the official, campus-wide system of record for people data was our locally developed Simon system. As each additional group or type of person came along, we had to identify the "system of record" for that part of the data feed.

Identifying systems of record is an iterative process. Once you identify a data element, and who in the organisation owns and maintains it, you then need to get them to agree to supply you with an on-going feed. This may be complicated by then discovering that the authority doesn't really have all the data you need. For example, we were finally able to identify the authority for building names, by asking the University President at an open "town meeting". This approach may not endear you to the lucky selectee, but it does help the process move forward. We then discovered that although we could get the "official" names for buildings, they didn't have the more common names[5] or abbreviations for buildings. Abbreviations are important – we cut the size of our campus directory from 220 pages to 203 simply by using building abbreviations! We are still working on resolving this issue completely.

Our current strategy for picking up the smaller groups and data elements, is when someone asks for it, we ask them who is authoritative for that group. For example, when the Provosts office asked for a mailing list for emeritus faculty, I asked them who was in charge of that information. They admitted that they

[1]Although I am starting from a university environment, we are also a major employer, with an active HR department. I am not going to address the ISP world.

[2]Such as contractors, vendors, adjunct faculty, visiting scholars, temporary employees, retires, conference guests, research collaborators, emeritus faculty, consultants, Ice Hockey officials, model railroad club members, etc.

[3]Stake holders included Parking and Transportation, Public Safety, Human Resources, Physical Plant, Contracts and Grants, the Student Union, the Registrar, the Provost, the Library, Sodexho (food service), Purchasing, and a number of others.

[4]SunGard SCT – http://www.sct.comhttp://www.sct.com .

[5]My office is located in the "Alan M Voorhees Computing Center," which everyone refers to as the "Voorhees Computing Center" or the "VCC".

were responsible, at which point I offered them a tool to maintain that list, that would also be able to give them mailing lists, interface with the campus mail room distribution system and feed into the ID card system. They agreed, we did some training and that part of the puzzle dropped into place. It was helpful to have a tool ready to go when they asked.

Sharp Edges

Even with a good data feed, you still have to watch out for some differing expectations between the users of the system and the mechanical reality. This issue was driven home recently at the end of the fiscal year. From a database standpoint, a data value is often a time/date value. If you have a time component, there is almost 24 hours between a termination date of 06/30/05 and a start date of 07/01/05. Thus, we had a bunch of folks who terminated on June 30th just after midnight and their new position did not start until the following day at midnight. As a result, about 40 people lost their parking access when they shouldn't have. We will be implementing a "look ahead" function that when someone's job status is terminated, we will look ahead to see if they are starting a new job within the next two weeks.

Empowering the Process

Although we had identified Banner as the system of record for employee data, and the Human Resources department as the maintainer of that data, we discovered that although Banner does a fine job tracking eligible employees and payroll, it wasn't ready to handle real time operations. From a database perspective, a potential employee is entered into Banner with some demographic information, a department affiliation, etc. This can take place weeks before their actual start date, and not everyone entered here actually starts as an employee. So rather than key off of this, we would wait until payroll actually entered them and assigned them a job and job classification. When all this was doing was driving the telephone directory, we could tell folks to wait a few days and the new folks would show up in the online directory. But when this entry was required to issue the person an ID card and a Parking transponder, waiting a few days was not acceptable.

Another requirement for employment, was the completion of the IRS I9 form. Historically HR had problems getting folks to come up to their office, which was located some distance from the main campus, to actually sign these forms. So we wrote a little application for HR staff, and made a deal with them. If they would use this application to mark when a new employee was "OK", (and indicate faculty or staff status), we would refuse to issue an ID card, or parking transponder or computer account until HR set the flag. With this in place, new employees had to visit Human Resources before they could really do much of anything. This also encourages the departments to follow the HR hiring process.

The other end of the employment process is employee separation. This impacts not only the IT world, but many other departments. When an employee leaves, the key shop needs to collect their keys; if the person was responsible for hazardous materials (gas cylinders, chemicals, etc.), someone else needs to take custody of it. In addition, separation will impact benefits such as pension and health care. One recurring problem we had seen, was when departments failed to renew employees on fixed term appointments, and essentially, their job ran out. In order to deal with this and other issues, Human Resources formed a "Separation Process" committee. One of the outcomes of that process was an automatic process to generate notification to departments of who was coming up for separation. By connecting with this process at the database level, we are able to synchronise our electronic world with the "official" employment status. Now if a department ignores these separation reports, a lot of things happen automatically. This process gave us a better employee data feed, and puts more teeth in the HR process.

Another example of cooperative projects is the management of Emeriti information mentioned above. The Provosts office agreed to maintain the lists, which feed into the ID card and directory systems. The tool they use also lets them get mailing lists back, and even the ID card photos.

One of our remaining challenges, is trying to eliminate or at least reduce the number of "Director Specials". Sometimes when a new person comes on board, "helpful" people high up in the food chain would try to grease the skids by arranging for the new person's email account to be ready when they show up. In the past, they would call their favourite director in the IT division, and get them to set up special computer account. This would sometimes result in a second account being created once the normal HR process went through, to make matters worse, this second account would be the official account, included in the department mailing lists, and so on. With wide scale use of card readers and parking passes, even if the person got their email account, they still couldn't get an ID card, get parking etc. While we might have to jump when the CIO says frog, the parking office doesn't! We are gradually convincing these directors and VPs, that the best way to get a new person on board, is to get the proper paperwork to HR, and from then on, the process is automatic and painless. We have on occasion walked the paper through HR, and the other systems, but even that extra effort is much less work than trying to unsnarl manual entries that entered the system in the middle.

Data Sources

One of our goals in the design and evolution of our system, was to let other people and departments do as much of the data entry and management as possible. Ideally, they would already be doing this work,

and we would just be able to tap into their systems. Although they may have been hesitant at first, the other departments seem to like this approach, if for no other reason, is that it makes them clearly the steward of their own data and individuals are encouraged to go to the "right place" to get their records and status cleaned up. Some obsolete fields have been omitted from the table descriptions that follow.

General People Schema

The key table for handling people in our system is aptly named PEOPLE(Figure 1). It has evolved over the years, with new fields being added and others being made obsolete, or on their way to being phased out. As we refine our data model, this table is moving back to it's intended role of identifying people, and other information is being moved to other tables.

The PEOPLE table shows it's heritage as the driver for creating computer accounts, and some of the data models of earlier systems used to feed it. One of the annoying lacking, is the the absence of a Middle_Name field. When upstream systems started providing a middle name as a unique field, it was automatically merged with the first name to be stored here. Later on, a Preferred_First_Name field was provided in another table, which can be set by the individual for use in directories and displays. There is a lot of code that know the old way of doing things, and fixing this will be a non trivial task (and is waiting for some other things to be rewritten).

Another problem we have, is with the Student and Employee flags. These are[6] used to control the creation and expiration of computer accounts. There have been special cases where someone needed an account, and one of these flags have been set via other means. However, we have run into cases where staff in the field see these flags set and think that the person in question is a current employee or student. I look forward to the day when these flags are gone and we rely on the status values.

The xxx_UID fields really don't belong here, and they do imply some policy, like restricting people to just one student and one employee account, but also to just one guest account. In practice, a student or employee should NEVER have a guest account, and there are cases where someone may need more than one guest account, or these accounts should not be re-used, such as for a contractor who works for one department, leaves, and returns to work for another department.

In cases where the person in question comes from Banner (which is all students and employees), we include their "PIDM" – this is the primary key used by Banner. The general practice is that once a person is in Banner, all of their identifying information (name, ID Number, SSN, DOB and Gender) comes from Banner. The inclusion of the social security number is disturbing to some people, but it has proven very useful at the ID desk to identify people who are returning, and are already in the system from those who are truly new to Rensselaer.

One of the ongoing challenges of maintaining the data, is dealing with people who get entered into the

[6]We will be changing this, and using status values instead.

ID	Number	Primary Key used to identify a person (or entity). Referenced by many other tables.
Lastname	Varchar2(60)	The family name or last name.
Firstnames	Varchar2(32)	First and Middle Names.
Prefix	Varchar2(8)	Prefix for a name – generally unused.
Suffix	Varchar2(8)	Formal suffix to name, such as "Jr", etc.
Student	Varchar2(1)	Indicates that person gets a student account – being phased out.
Employee	Varchar2(1)	Indicates that the person gets an employee account – being phased out.
Guest	Varchar2(1)	Indicates that the person gets a guest account – being phased out.
Student_Uid	Number	The unixuid of the student account, if any.
Employee_Uid	Number	Unixuid of the employee account, if any.
Guest_Uid	Number	Unixuid of the guest account, if any.
ISO_Number	Number	ISO Format ABA card number – ID card number.
FAIMS_PIDM	Number	Primary key for person identification in Banner.
Spriden_Id	Varchar2(9)	University ID number (Rensselaer ID Number – RIN) – assigned in Banner.
Spriden_Activity_Date	Date	Date of the last activity in the person ID table in Banner.
Budget_Str	Varchar2(32)	Default budget number for non student charges.
SSN_Str	Varchar2(9)	Person's social security number.
Birth_Date	Date	Date of Birth.
Gender	varchar2(1)	Gendor of person.
Clean_Lastname	Varchar2(64)	A "cleaned" version of the lastname (all lowercase, with spaces and punctuation removed) to aid in searching.
Clean_Firstnames	varchar2(32)	A cleaned version of the firstname.

Figure 1: People description.

system twice. We have a merge tool that helps us shuffle records between the different versions of a person, and marking the "bad" record so it is not re-used.

Students, Employees

For student data, we have a process that compares the Registrar's list of students with our own copy of the student list. Along with daily runs, we can update a specific student's records via a web application. We had the added challenge, in that we actually have (or had) two registrars, one for each of our campuses, and they had different practices in marking "active" students. This difference is demonstrated in the discussion of the Person_Status view below. Student processing is fully automatic now. Student with account, status or ID card issues are referred to the Registrar. The Registrar also notifies us when they recode large groups of students, so we don't panic when 1200 students drop out suddenly (due to graduation.)

In a similar manner, employee data is checked once a day with the Human Resources information in Banner. We can also do updates of a specific individual via a web tool. The Banner data for employees isn't as clean as we would like, which required a special tool for HR to use to indicate when employees really start. Fortunately, this process has provided some benefits to HR, that they are quite willing to use the tool (described previously). We did add a safety check that stops employee processing if more than 10 percent of the employees are marked as being changed.

Non-Traditional Sources

There are some groups of people, such as emeriti and retirees who are already in the system, but no longer active employees. Since we knew that these people were already in our system, all we needed to do was maintain a list of them. As part of our directory project, we had a tool available to manage mailing lists. Some minor changes gave us a version of the tool to maintain lists of emeriti and retirees. These mailing lists were then fed back into the system to provide additional status entries.

Guests

Although we had students and employee status well in hand, that still left "none of the above." With the deployment of RFID cards and Parking transponders to provide access to campus buildings and parking facilities, the problem of maintaining status information for "ID Guests" became very real. Our original practice of creating a campus computing account (which as a side effect, would generate an ID card number for them) was not going to handle this.

One of the principles that we adopted, the 13th amendment to the Constitution notwithstanding, is that everyone had to be "owned" by an on campus person or department. For regular employees, Human Resources plays this role and for students, the Registrar handles it. But for everyone else, we needed to designate an "owner". In many cases we were able to use our existing departmental directory administrators to handle this role, and for other cases we created new "departments" and assigned administrators there. These administrators are assumed to know which guests are still with their department and can expire or renew them as needed. For some categories of visitors, they can also control if that person gets a computer account and if that person is to be included in the online directory.

The primary point of contact for guests, is our ID card office. The folks there have a tool that lets them make entries in the ID_People table (Figure 2). Entries here are automatically propagated into the People table, and are also used to provide status values. Some of the fields have been omitted for brevity including some personal ID fields (SSN, DOB, etc.), but the key ones are included.

Some entries in the ID_People table will have a Sponsor specified. The sponsor will point to another person, and this used used in the case of dependents or spouse, or a personal care worker; someone whose relationship with Rensselaer is a direct result of some other member of the community. This allows us to automatically "expire" dependents when their sponsor is no longer eligible to sponsor dependents.

If an entry doesn't have a sponsor, then it will have an Affiliation_Id. This points to a department. Originally, the department tree for the ID Guest system and the department tree for directory were different. We have since merged them and although we still have two tables, the tools keep them synchronised. One of the big benefits of merging these two trees into one, is that the directory departmental administrators can maintain ID guests as well.

Person_Id	Number	Primary key – matches People.Id
Lastname	Varchar2(64)	The last name.
Firstnames	Varchar2(32)	The first and middle names.
SSN	Number	The SSN or Rensselaer ID Number if available.
Entry_Type_Id	Number	Identifies different types of entries, see Id_Entry_Types.
Sponsor	Number	The People.Id of the person who "sponsored" this guest, if any.
Requestor	Number	The People.Id of the person who requested this entry.
Affiliation_Id	Number	An identifier of the department sponsoring this entry.
Expiration_Date	Date	Date this guest entry expires (can be renewed).
Dir_Flag	varchar2(1)	Flag to control inclusion in the campus directory.

Figure 2: ID_People description.

The ID Card Office staff use the tool to enter a new person into the system. The application attempts to locate the new person in the existing database, searching by name and by ID number. We really want to avoid duplicating people. We can still make the guest entry, but the Person_Id will match the original record. The next step is to select what type of guest they are, which then determines the additional questions they need to answer. The choices are contained in Id_Entry_Types (Figure 3). The xxx_Required flags tell the application if it needs to prompt for those fields. In the cases of sponsor and requestor, the person must be in the existing database, and in the case of a sponsor, the sponsor's status must be one that allows them to sponsor a guest. (How this is done will be discussed later). If the entry type has an affiliation id, that tells the application the root of the departmental tree it should use for the selection list. This ensures that Incubator staff must work for an Incubator company and so on. The department administration tool will expire all of the members of that department (or other organisation) is terminated. Our current list of entry types is in Figure 4.

Other flags will determine further processing. The directory flag can put someone into the directory automatically with no choice to the departmental administrator. For example, our ROTC staff go in – no choice. Other categories such as visiting researchers or Incubator staff can go in, but that can be controlled by the departmental administrators ("O" optional, default include, "P" – optional, but default to not include) and the rest don't go into the directory at all. If there is an expiration delta, that number of days is added to the current date to get a default expiration date.

Entry Points, Delegation, Short Term Visitors

Not everyone on campus is there long enough to get their own picture ID card. While they still need one for building access and dining plans, the overhead of producing ID cards for someone who was going to be on campus for three days for a conference was simply not worth it. There are a few cases where an ID card is desirable as a keep-sake, for the most part, we have a collection of generic cards (guest1-Guest999).

While that worked for the physical access, we still had the problem of visitors to campus who needed to authenticate to access our VPN to get on campus, or to access the wireless network. This would result in the request and creation of a full computer guest account, with email access, printing allocations, disk charges, and would take a business day or two to get set up, or someone in the department would "loan" their account out. While the first option was arduous,

Entry_Type	Varchar2(24)	A short name for this type of entry.
Sponsor_Required	Varchar2(1)	A flag indicating of a sponsor is required for this type of entry.
Requestor_Required	Varchar2(1)	A flag indicating of a requestor is required for this type of entry.
SSN_Required	varchar2(1)	A flag indicating if an ID number is required.
Affiliation_Id	Number	An identifier of the department sponsoring this entry.
Expiration_Delta	Number	Number of days from now for entries to expire.
Dir_Flag	varchar2(1)	Flag to help control inclusion in the campus directory.
Display_Rank	number(3)	A rank order to be used for display purposes.

Figure 3: ID_Entry_Type description.

Entry Type	Exp Delta	Sponsor Required	Requestor Required	Dir Flag	Display Rank	Entry Type Id
Special Programs	60	N	Y		100	91121096
Temporary Employee	90	N	Y	P	100	91318569
Conference Guest	60	N	Y	N	100	91114523
Off Campus Contractor	120	N	Y		100	91393994
On Campus Vendor	120	N	Y	P	100	91398702
Department Vendor	90	N	Y	P	100	91402506
RU Club Member	180	N	Y		100	91405098
Incubator	180	N	Y	P	100	91068656
Tech Park	180	N	Y	P	101	91080046
Dependent		Y	N	N	160	91449911
Spouse		Y	N	N	160	91449912
Domestic Partner		Y	N	N	160	91449913
Personal Aid		Y	N		200	91449917
Visiting Researcher	120	N	Y	O	300	91399849
Visiting Faculty	120	N	Y	O	300	91399850
ROTC Faculty	90	N	Y	Y	400	91397007
ROTC Staff	90	N	Y	Y	400	91397008

Figure 4: Current ID Entry types.

the second could result in the staff member who "loaned" the account being terminated.

Instead, we developed a tool that allows departmental administrators (and each department has at least one, see the next section) to allocate a temporary account that will allow someone to access the VPN to get in, or the wireless network to get out. When they allocate an account, they can specify an expiration time from 3 hours up to 2 days (accounts can be renewed), after which time, the password will be reset automatically. They can also specify a comment for that assignment that they can later review. Once allocated, these accounts "belong" to that department. Once they have expired, they are available for re-use by that department. If the department has no "free" accounts, a new one will be drawn from a pool. The pool is monitored to ensure a ready supply of new temporary accounts. All of this happens without any involvement by the IT staff – it is entirely in the hands of the department administrators.

This gives departmental administrators the ability to satisfy some of the needs of visitors to their department quickly and easily.

Departments

We really can't talk too much about managing people, without understanding what department they belong to. One of the key points of this system, is that there is someone responsible for maintaining the status information for everyone on campus. This may be indirect like with HR maintaining employee information, or the Registrar maintaining student information, or directly with departmental administrators maintaining guest information. Although there are a few categories such as dependents or spouses that are "owned" by another person, all of the rest of the guests need to be "owned" by a department. This requires a good list of departments, and possibly the relationship between departments (all of the engineering departments belong to the school of engineering, etc.).

As part of the telephone directory project, we had to get a pretty good handle on departments. When the University went to a fund accounting system with composite account numbers, one of the elements was the ORGN (Organisation). These were arranged in a hierarchy, with every orgn rolling up to it's parent and so on to the president. Our joy was short lived however. The primary purpose of the ORGN tree, was financial accounting. This resulted in lots of extra "departments" (like copy center), departments that were not "data enterable", so these were paired with an "office of XYZ" department so charges could be rolled up. To make matters worse, the department "names" were limited to 30 characters, and there was no place, nor interest from the finance office in maintaining abbreviations.

To get around this, we created our own superset of the ORGN tree, where we could add our own "virtual" organisations, specify alternate names for real organisations as well as abbreviations. An agreement was made that we would only create ORGN codes (the primary key) that started with "V", "S" or "DH", and the rest of the name space belonged to them. New ORGNs from the controller's office are automatically added to the directory (and the telecom staff double checks them – some are not included). But this gave us a place to expand the existing university departmental hierarchy, as well as build new trees for vendors, and many other special groups.

Another aspect of the department management, was to identify one or more administrators for each department. If a department did not have one, it could inherit administrators from it's parent. Although this was originally intended for managing the telephone directory, it gave is a ready made list of people who can take on additional responsibility for guests in their department. This was a key component in ensuring that status information for EVERY person on campus was the responsibility for an identified person (or department) to maintain.

People Status

Now that we have identified all of our different data feeds, it is now time to put everything together into a uniform object (Figure 5). The first step is to create an Oracle view, Person_Status (Figure 6) that combines information from Employees, Students, ID_Guests, Directory_Aux_Entries and Hartford_Raw_Dir

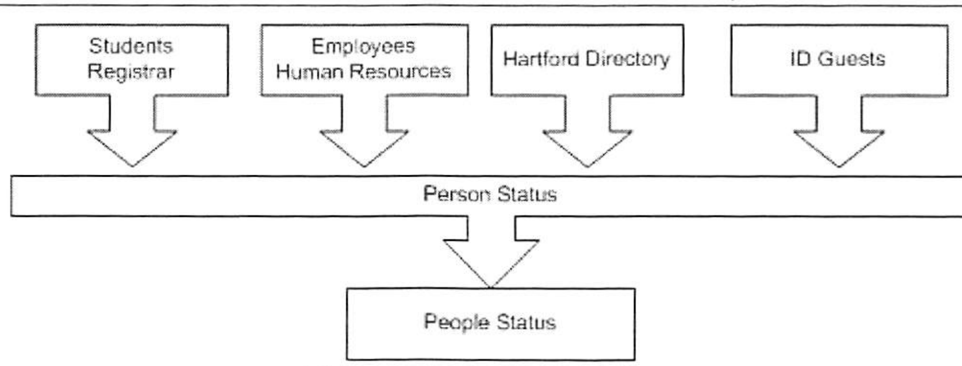


Figure 5: People Status data flow.

tables. We don't want to be querying this view all of the time, and we do want to know when someone's status changes, and maintain some history. To this end, we have a People_Status table (Figure 8). And finally, we feed this information to other systems.

Person_Status

The person status support was originally developed to provide a patron feed for our Library circulation system. It was later enhanced to feed our ID card system. It is now being generalised as people status management tool, however, some traces of the original library and ID card system remain in the column definitions. Most of this information is being moved to other tables that join with Person_Status based on the Status_ID column. The Person_Status view (Figure 6) is the key place where all the different inputs to the status process are brought together.

The Person_Status view is coupled tightly to People_Status_Types (Figure 7) to generate a snapshot of everyone's "status". The view and this table is where all of the site specific stuff lives. The view goes and roots around in a number of different database tables to come up with a standard set of status entries. The People_Status_Types table has columns for joining with the view to determine types, and some other columns that are used to feed the Library and some other systems. I have omitted this second set of columns, as they are gradually being phased out in favour of other tables. A subset of the entries in the People_Status_Types is available in the first appendix of the paper.

The Person_Status view is a series of joins with other tables and the People_Status_Types table, that are connected together with a SQL UNION directive. For ease of formatting, I have broken the view up into individual stanzas (Displays). This is where all the magic happens.

Get Employees

This first section looks in the employees table to get our current employees. The important distinction here is really the key 1 looking for "A" (active) employees and

Lib_Patron_Type	varchar2(3)	Patron Type for Library Circulation System
Status_Id	Number	Identifier for status type.
ID_Card_Status	varchar2(32)	"Status" entry for use on the ID card.
Person_Id	Number	Identifier for the person record for this entry.
Orgn_Name	Varchar2(32)	Name of the department, if applicable.
Orgn_Code	Varchar2(6)	Department Identification code. Used with Coas_Code to identify department.
Coas_Code	Varchar2(1)	Department Chart of Accounts identifier.
Type_Key	Number	Affiliation identifier for non departmental entries.
In_Dir	varchar2(1)	Flag indicating if person should be in the fac/staff directory.
End_Date	Date	Date when status expires, if known.

Figure 6: Person_Status description.

Status_Id	Number	Identifier for status type.
Rank	Number	Orders status values, highest value is returned.
ID_Card_Status	varchar2(32)	"Status" entry for use on the ID card.
Person_Id	Number	Identifier for the person record for this entry.
Status_Category	varchar2(8)	Rough category for status types.
Source_Table_Name	varchar2(32)	The name of the table we will match with.
Key_1	varchar2(16)	A key to match against, usage depends on category and table name.
Key_2	varchar2(8)	A second key (there is also a key_3 and key_4).
Nkey_1	number	A numeric key, usage depends on the status category and source table.
Comments	Varchar2(255)	A description of this entry.

Figure 7: People_Status_Types description.

```
select lib_patron_Type, status_id, id_card_status, person_id,
       substr(Ftvorgn_Title,1,32), pebempl_orgn_home,'9',
       to_number(null) TYPE_KEY, pst.in_dir, e.nbrjobs_end_date
  from employees E, people_status_types pst, fimsmgr.ftvorgn f
 where Status_Category='Emp' and key_1 = NBRJOBS_STATUS
   and PEBEMPL_ECLS_Code like nvl(key_2,PEBEMPL_ECLS_Code)
   and pebempl_orgn_home = f.ftvorgn_orgn_code
Union...
```

Display 1: Retrieving employees.

key 2 looking for the employee classification. It also dives into the Ftvorgn table to come up with the department name.[7] See Display 1.

Get New Employees

We dive back into the employees table again to look for new employees. The employee classification doesn't appear in the employees table until the next payroll cycle, and we really don't want to wait that long to start issuing ID cards and the like. So Human Resources has a tool to set the HR_Ok and HR_Status fields appropriately, and this stanza will pick those up. In the course of normal employee record processing, these fields will get cleared when the classification information comes through so we don't have people with two employee status values. But even if something goes wrong, and that isn't cleared, since the regular employee status types have a higher rank, that entry will override this entry. See Display 2.

Get Troy Students

Continuing with the view, we go after our local students. We are getting the same number of columns as we did in the first stanza (a requirement of a view), but most of the department related fields we are returning as null. We look for active students Sgbstdtn_Status_Code='AS', and those in the Troy campus (Key 1), and also breaking them down by College (school) and level (Grad/Undergrad). Although this level of granularity may be overkill for most applications, the Library wanted this fine grain distinction. See Display 3.

Get Hartford Students

We have a second campus in Hartford, and they code students somewhat differently. In Troy, the Registrar aggressively recodes students who are not active. But our Hartford campus does a lot of continuing education, where students just take a course or two, so their status is left as 'AS'. So to handle this case, we pull the same columns, but look for some other keys in the students table, Registered_This_Term. We actually have a third campus which is being phased out, so I will skip that stanza of the view. See Display 4.

Mailing List Entries

Our method of handling people who are already in the system, and need some special status, like what we did for the Emeriti faculty, is handled internally by adding them to a special "department". These tools were originally developed for the phone directory, and worked well here as well. In this case, an entry is made in the Directory_Aux_Entries table (which is listed as Dir_Aux_Ent in the appendix), and we pick up the appropriate entries.

[7] People well versed in Oracle and/or Banner will note that column aliases are missing, and additional conditions are needed to make this work. These have been removed to aid formatting. See the end of the paper for how to view the actual, working code.

```
select lib_patron_Type, status_id, id_card_status, person_id,
       nvl(substr(Ftvorgn_Title,1,32),'Unknown'), pebempl_orgn_home, '9',
       to_number(null), pst.in_dir, to_date(null)
  from employees E, people_status_types pst, ftvorgn_coas_9 f
 where Status_Category='NewEmp'
   and hr_ok = 'Y' and hr_status = key_1
   and nvl(pebempl_orgn_home,'XXXXXXX') = f.ftvorgn_orgn_code (+)
Union...
```

Display 2: Select new employees.

```
select lib_patron_Type, status_id, id_card_status, person_id,
       Sgbstdn_Majr_Code_1, Null, Null, to_number(null), Null, To_Date(Null)
  from banner_students bs, people_status_types pst
 where Status_Category='BStu'
   and Key_1 = SgbStdn_Campus_Code
   and ( key_2 = Sgbstdn_Coll_Code_1 or key_2 is null )
   and Key_3 = Sgbstdn_Level_Code
   and Sgbstdn_Status_Code = 'AS'
Union...
```

Display 3: Select Troy students.

```
select lib_patron_Type, status_id, id_card_status, person_id,
       Sgbstdn_Majr_Code_1, Null, Null, to_number(null), Null, To_Date(Null)
  from banner_students bs, people_status_types pst
 where Status_Category='HStu'
   and Key_1 = Sgbstdn_Level_Code
   and ( Hart_Reg_This_Term = 'Y' or Registered_This_Term = 'Y' )
   and Sgbstdn_Campus_Code = 'H'
   and Key_2 = Sgbstdn_Status_Code
Union...
```

Display 4: Select Hartford students.

We use a different table for the department name this time, this stanza was written later and I should revise the employee entries. See Display 5.

ID Guests

This is the place where we get all of our id guests. A number of the id entry types map directly to a status type. You can actually compare the Nkey_1 values in the appendix with the Entry_Type_Id values in the Entry Types listing above. We also make a check to ensure that the guest entry either does not have an expiration date, or that date is some time in the future. If there isn't one on the record, the current time will be used, but that is good enough! See Display 6.

People Living on Campus

We have a significant number of people who are living on campus after they graduate. One of the side effects of graduating, is losing your student status, which triggers a number of events, including cancellation of your ID card, which locks you out of your building! Although these people were no longer registered students, they did have housing contracts. We added some information about room assignments to our student table, and added this final stanza to the Person_Status view. During the normal school year, this entry will provide a second status value to all on campus students, but since it is of a lower rank, it is ignored. See Display 7.

People_Status

While the Person_Status view will give us a snapshot of the current status of everyone, it is not able to provide any historical information. A common question when checking someone with no status, is what was their status before it ended. There were also concerns about performance of this view – it is looking into a number of other database tables. For both of those reasons, we have a process that runs daily that checks the current status of everyone, with what we have saved in the People_Status table (Figure 8).[8] We can also invoke this processing for a specific person. This function is used for several administrative applications when they want to ensure someone's status record is current as of this instant. Refreshing a single person's status is almost instant, it takes longer to redraw the screen.

The People_Status table has the same columns as the Person_Status view, with the addition of a start and end date, as well as some numeric columns (When_Inserted and When_Marked_For_Delete) to assist with data propagation. Changes in a person's status are also reported in the Meta Change Queue subsystem, so other systems can easily watch for changes in a person's status.

[8]On a recent run, this process compared 19,000 records in just over a minute, a performance level I am comfortable with.

```
Select lib_patron_type, status_id, id_card_status, person_id,
       substr(nvl(orgn_common_name,orgn_name),1,32), dae.orgn_code,
       dae.coas_code, title_id, dd.def_include, to_date(null)
  from Directory_Aux_Entries DAE, people_status_types pst,
       directory_departments dd
 where Status_Category = 'DirAux'
   and pst.Key_1 = DAE.Orgn_Code and pst.Key_2 = DAE.Coas_Code
   and dae.orgn_code = dd.orgn_code and dae.coas_code = dd.coas_code
 Union..
```

Display 5: Select mailing list entries.

```
select lib_patron_Type, status_id, id_card_status, person_id,
       ia.name, ia.orgn_code, ia.coas_code,
       nvl(ip.affiliation_id,ip.sponsor), ip.dir_flag,
       ip.expiration_date
  from id_people ip, people_status_types pst, simon.id_affiliates ia
 where Source_Table_Name = 'Id_People'
   and Nkey_1 = Ip.ENTRY_TYPE_ID
   and nvl(ip.expiration_date,sysdate) >= sysdate
   and Key_1 = 'CURRENT'
   and ip.affiliation_id = ia.affiliation_id (+)
Union...
```

Display 6: Select ID guests.

```
select lib_patron_Type,status_id,id_card_status,person_id,
       Slrrasg_Bldg_Code || '-' || Slrrasg_Room_Number,
       Null, Null, to_number(null), Null, Slrrasg_End_Date
  from banner_students bs, people_status_types pst
 where Status_Category='ResLife'
   and Slrrasg_Active = 'Y'
   and ( key_1 is null or key_1 = slrrasg_bldg_code )
```

Display 7: Select people living on campus.

Making Use of People Status

The original target of the people status support was to feed the circulation system for our Library. It was later expanded to feed the ID card system. It became obvious that extending the People_Status_Types table for every new application was the wrong approach. Instead, we added a set of People_Status_Aux... tables that allowed us to define new flags for each status type, and developed a tool to allow administrators of other systems to set their flags and not interfere with other consumers of the status information. We also use a person's status to trigger other processing.

People Status Auxiliary

Adding columns to tables, or creating new tables, and then writing interface routines and tools to manage them can get pretty tedious, and takes a lot of time that could be better spent on other tasks. I wanted to be able to "extend" status related function by making table entries in the database, rather than writing new code.

To start, we defined the table People_Status_Aux_Master (Figure 9) that define new "streams" of status information. Initially I was envisioning selecting people for data feeds (or streams) to other systems, and the name stuck. This defines the name of the stream and lists an Oracle role that will be used for access control by the administrative tool.

The next step was to define a prototype table, People_Status_Aux_Proto (Figure 10), to identify the possible fields, the field types and the default values. This is used by the administrative tool to automatically generate the appropriate columns and switches on the web page.

The final table holds the actual values for each stream, People_Status_Aux_Values (Figure 11) which stores a field value for each status, stream, field triple.

Once we have defined a new stream, assigned it an access role, and defined one or more fields for that stream, we are able to delegate the management of these values to the appropriate interested parties. At present we have the following streams defined and in use; see Figure 12.

Now that we have all of this information in the system, and control delegated to the appropriate offices, we need to get it out again. At the most basic level, we have a PL/SQL package with some routines, one that will give you a list of everyone with a particular Stream/Field/Value triple and another that will return true or false, if a given person has a particular

Person_Id	Number	Person identifier.
Status_Id	Number	Identifier for status type.
Start_Date	Date	Date when this status was initiated.
End_Date	Date	Date when this status will terminate if known or was terminated.
Orgn_Name	Varchar2(32)	Name of the department, if applicable.
Orgn_Code	Varchar2(6)	Department Identification code. Used with Coas_Code to identify department.
Coas_Code	Varchar2(1)	Department Chart of Accounts identifier.
Type_Key	Number	Affiliation identifier for non departmental entries.
In_Dir	varchar2(1)	Flag indicating if person should be in the fac/staff directory.
When_Inserted	Number	Sequence value when record was inserted.
When_Marked_For_Delete	Number	Sequence value when record was considered to be deleted.

Figure 8: People_Status description.

Stream_Name	varchar2(32)	The name of the stream.
Access_Role	varchar2(32)	An Oracle role that can manage this stream.

Figure 9: People_Status_Aux_Master description.

Stream_Name	varchar2(32)	The name of the stream. Must exist in the Master table.
Field_Name	varchar2(32)	The name of the field.
Field_Type	varchar2(32)	The data type – to assist the web tool in formatting and controlling.
Field_Length	varchar2(32)	The maximum length of the field where applicable.
Field_Default	varchar2(32)	The default value to use.
Field_Rank	Number	Rank order to display fields on the web tool.

Figure 10: People_Status_Aux_Proto description.

Status_Id	Number	The People_Status_Types.Status_Id of the status getting this value.
Stream_Name	varchar2(32)	The name of the stream.
Field_Name	varchar2(32)	The name of the field.
Field_Value	varchar2(32)	The value for this particular field.

Figure 11: People_Status_Aux_Values description.

Stream/Field/Value value. We have also set up Generate_File [6, 8] targets to extract list based on stream and field combinations. We have also written several "wrapper" packages to provide the appropriate streams to people and applications who need to connect directly to the database.

Other Uses

Our physical access control system (Card readers, parking gates), also uses status, or more specifically the lack of any status to automatically terminate all access to campus building, roadways and parking. We also use status to control if and how people are included in the different campus directories.

Conclusions

The people status project has been evolving over a number of years here. During that time, some of the original data sources have been replaced, and new ones have been added. In some cases the direction of data flow has changed. At one time the faculty/staff directory data drove the status information; that has been reversed, status data now drives the faculty staff directory.

The People Status project at Rensselaer has reached critical mass. It impacts enough systems that matter to people (such as Parking!) that folks are willing to play by our rules. And it does make life simpler – students enter the system via the Registrar, employees get in via Human Resources and everyone else comes in via the ID card office. Once in, the assorted special cases get picked up by the departments who care about those people and things work well and with a minimum of human intervention.

Implementation Lessons

The people status project was not rolled out as a complete package starting at nothing, but rather was built on several smaller projects such as the directory and computer account management. These systems allowed us to identify and refine the data sources and procedures, while assembling the infrastructure. It also provides time and opportunity to develop the working relationships with key players in other departments.

There are two types of people I needed to work with to pull this project together, people who had data I could use, and people who needed data. By starting with some smaller projects (they didn't seem small at the time!) like the phone directory and computer account management, I was able to put together a comprehensive enough "people" database, that I could get people who needed feeds interested. By having tools and techniques readily available to enable them to control their feeds (Generate File, Meta Change Queue, etc.), I was able to get a number of "clients" for "my data". These clients in turn made it more attractive for the owners of the data to work with me, since it would leverage their work.

I was also able to target groups that were outside of the mainstream, and so didn't have quite the level of IT support that they may have wanted. I help them, and they are willing to modify their procedures and processes to better accommodate what I need. I also find it very helpful to visit my "clients", and see what they are doing.[9] I have been able to provide them with a tool that greatly simplifies their operation (often times, it was an existing tool that needed slight modification) and I strengthen a person to person relationship as a result.

One important lesson, is that your source data is often not quite what you need – It may be too early or too late (like with HR) requiring some extra tools to make it useable, or intended for a different purpose and it will need to be adapted and cleaned up (like with the departmental tree). You have to work with your sources to make it work. The solution may be with people and process and not technology.

Systems and Processes Impacted By This Project

A number of systems and business processes (see Figure 13) has been impacted and improved by this project. The most common improvement is the automatic removal (or at least notification) when people leave or their status changes.

[9]Is this an obvious part of customer service – Yes! Is it often overlooked, sadly, Yes.

EBS List	For important campus announcements, we have email lists like "All Faculty", "All Staff", "All Students" that are maintained based on current status. Our Postmaster keeps these mappings in place.
Hostmaster	For our host database, there are requirements as to who can "own" or "administrate" machines on our campus network. Our Hostmaster makes this determination.
ID Access Control	We have several buildings that are open to the campus "community" via card readers. This enables our Access control staff to control that definition of community.
ID Card Admin	There are two controls here, one to indicate if that person's information is maintained in Banner (if not, the ID desk can make the updates directly), and also if that person is eligible to have a dependent.
Package Tracking	Our campus mail room wants an address feed, but not of everything. They pick and choose.
PC Store	Our Campus Computer store is limited in who they can sell to (due to contracts with vendors). This helps them identify who is ok, and who is eligible to charge their purchases to a student account.

Figure 12: Current auxiliary streams in use.

A Note on Table Definitions

There are a number of Oracle table definitions included in this paper. In order to save space and assist with formatting, I did not include a number of columns that are in most of the Simon tables. A recurring set of columns that we see in many of the Simon tables are When_Inserted, When_Updated and When_Marked_For_Delete columns. These are filled with an ever growing sequence, and are used to identify records that have changed since some previous point in time, and to propagate those changes to other tables and systems [4]. Many tables also have a Clerk or Clerk_Id field to indicate who touched that record last, and some also have an Activity_Date column to indicate in a more human form, of when that change took place. I have also removed references to those columns in some of the source code examples included here. The full definitions of both table and source code can be obtained via the Web. See the following section for details.

Futures

We continue to identify new groups and categories of people that need status. Adding these new types has become pretty trivial, with the tools and techniques being quickly adaptable to new situations. One offshoot of this project deals with physical access control. We mapped the people status auxiliary values into a more general demographic mapping module (along with departmental affiliation, course registration, etc.) into consumers of group information such as the access control system, and windows protection groups. Now when a student registers for a particular class, not only do they get access (via their ID card), to the room with the lab equipment, but access (via their computer account) to restricted course files and directories on Windows and AFS file servers. There might even be another paper on this topic for next year.

References and Availability

This is not a comprehensive, stand alone, product that we can package up easily for distribution. Despite efforts over the years to move our business rules out of the code, and into data table and views, there is a lot of site specific stuff in here. That being said, I feel that there is a lot here that can be used by other sites. You will have to do some code development of your own to address your own site's specific issues and requirements. Some of the existing packages we have developed may serve as models for your own work. The web tools we have developed use a pretty much standard Oracle web environment with our own custom front end to handle the authentication.

All of the PL/SQL source code for the Simon system as well as the full table and view descriptions are available via the web at http://www.rpi.edu/campus/rpi/simon/misc/Tables/simon.Index.html and http://www.rpi.edu/campus/rpi/simon/misc/Tables/SIS-index.html. If you ask nicely, I will try to answer questions and might be able to dig out some of the C and JAVA code that makes up other parts of the system.

Acknowledgements

I would like to thank Tom Perrin for his shepherding of this paper with me. I also want to thank Rob Kolstad for his excellent (as usual) job of typesetting this paper.

Author Biography

Jon Finke graduated from Rensselaer in 1983 with a BS-ECSE. After stints doing communications programming for PCs and later general networking development on the mainframe, he then inherited the Simon project, which has been his primary focus for the past 14 years. He is currently a Senior Systems Programmer in the Communication and Collaboration Technology department at Rensselaer, where he continues integrating Simon with the rest of the Institute information systems. In addition to the Simon project, Jon is also involved with the support of the Telecommunications billing system,[10] and providing data and interfaces for Unity Voice Messaging and CISCO VOIP deployment projects at Rensselaer. When not playing with computers, you can often find him merging a pair of adjacent row houses into one, or inventing new methods of double entry accounting as treasurer for Habitat for Humanity of Rensselaer County. Reach him via USMail at RPI; VCC 319; 110 8th St; Troy, NY 12180-3590. Reach him electronically at finkej@rpi.edu. Find out more via http://www.rpi.edu/~finkej.

[10] AXIS – Pinnacle CMS by Paetec.

Library	Circulation and Patron database – drives borrowing limits, aids with contacting via address feeds.
Access Control	Automatic revocation of access when a person leaves, automatic access for community members.
Computer Accounts	Automatic creation and expiration.
Human Resources	HR controls when a person will be issued access, email, etc. – ensuring compliance with employment rules.
ID Card Office	Single point of contact for guests, identification of departmental contacts, refinement and documentation of policies and procedures.
Directory	Status drives inclusion in the online and printed directories.

Figure 13: Systems and business processes impacted by this system.

Bibliography

[1] Anderson, Eric and Dave Patterson, "A retrospective on twelve years of LISA proceedings," *13th Administration Conference (LISA 1999)*, pp. 95-107, USENIX, November 1999.

[2] Arnold, Bob, "Accountworks: User create account on SQL, Notes, NT and UNIX," *The Twelfth Systems Administration Conference (LISA 98) Proceedings*, pp. 49-61, USENIX, December, 1998.

[3] Cooper, Michael A., "Spm: System for password management," *The 9th Systems Administration Conference (LISA IX) Proceedings*, pp. 149-170, USENIX, September, 1995.

[4] Finke, Jon, "Data propagation between oracle tables," *Proceedings of Community Workshop '92*, Troy, NY, June, 1992.

[5] Finke, Jon, "Institute white pages as a system administration problem," *The Tenth Systems Administration Conference (LISA 96) Proceedings*, pp. 233-240, USENIX, October, 1996.

[6] Finke, Jon, "An improved approach to generating configuration files from a database," *The Fourteenth Systems Administration Conference (LISA 2000)*, pages 29-38, USENIX, December, 2000.

[7] Finke, Jon, "Embracing and extending Windows 2000," *The Sixteenth Systems Administration Conference (LISA 2002)*, USENIX, November, 2002.

[8] Finke, Jon, "Generating configuration files: The director's cut," *The Seventeenth Systems Administration Conference (LISA 2003)*, pp. 195-204, USENIX, October, 2003.

[9] Finke, Jon, "Meta change queue: Tracking changes to people, places and things," *The Eighteenth Large Installation Systems Administration Conference (LISA 2004)*, pp. 231-239, USENIX, November, 2004.

[10] Harlander, Dr. Magnus, "Central system administration in a heterogeneous unix environment: Genuadmin," *USENIX Systems Administration (LISA VIII) Conference Proceedings*, pp. 1-8, USENIX, September, 1994.

[11] Hughes, Doug, "User-centric account management with heterogeneous password changing," *The Fourteenth Systems Administration Conference (LISA 2000)*, pp. 67-76, USENIX, December, 2000.

[12] Rosenstein, Mark A., Daniel E. Geer, Jr., and Peter J. Levine, "The Athena service management system," *USENIX Conference Proceedings*, pages 203-211, USENIX, Winter, 1988.

Appendix A: Selected People_Status_Types

RNK	ID Card Status	Status Category	Source Table	Key 1	Key 2	Nkey 1	Description
750	Employee	Emp	Employees	A	E%		Exempt Employees
750	Employee	Emp	Employees	A	X%		Executives
750	Employee	Emp	Employees	A	N%		Non Exempt employees
750	Faculty	Emp	Employees	A	F%		Faculty
740	Employee	NewEmp	Employees	Staff			Newly hired staff
740	Hartford Emp	HartEmp	HartfordRawDir				Hartford Employees
740	Faculty	NewEmp	Employees	Faculty			Newly hired faculty
730	ROTC Staff	N/A	id_people	CURRENT		91397008	US Military personal assigned to the ROTC detachments in a support (non teaching) assignment.
730	ROTC Faculty	N/A	id_people	CURRENT		91397007	US Military personal assigned to a ROTC detachment who will be teaching ROTC and other courses.
730	On Leave Employee	Emp	Employees	F			On Leave with full benefits
730	Employee (PT)	Emp	Employees	P			On Leave w/ Partial Pay and Benefits
650	Visiting Researcher	N/A	id_people	CURRENT		91399849	A person doing research (but not being paid by RPI)
650	Research Professor	N/A	id_people	CURRENT		91399850	Someone who is teaching, but is not paid by RPI. Will have a memo from the Provost's office or the Dean's office.
640	Undergrad	BStu	Banner_students	T	A		Undergrad, in Architecture at Troy
640	Graduate	BStu	banner_students	T	S		Graduate, in Science at Troy
640	Graduate	BStu	banner_students	T	E		Graduate, in Engineering at Troy
640	PDE Grad	DStu	banner_students	GR			Graduate Student with PDE
640	HartGrad	HStu	banner_students	GR	AS		Graduate Student at Hartford
640	Undergrad	BStu	banner_students	T	E		Undergrad, in Engineering at Troy
640	Undergrad	BStu	banner_students	T	S		Undergrad, in Science at Troy
640	Graduate	BStu	banner_students	T	A		Graduate, in Architecture at Troy
560	Incubator	Incubatr	id_people	CURRENT		91068656	A person affiliated with a company in the incubator center.
550	Dependent/Spouse	Dependnt	id_people	CURRENT		91067788	Dependent/Spouse of existing RPI person
550	Personal Services	N/A	id_people	CURRENT		91449917	A personal aid, generally for health care of a disabled member of the Rensselaer Community.
550	Family	N/A	id_people	CURRENT		91449912	A spouse of a member of the RPI community
550	Family	N/A	id_people	CURRENT		91449913	A domestic partner of a member of the RPI Community.
550	Family	N/A	id_people	CURRENT		91449911	A dependent, spouse or domestic partner
550	Temp Employee	TempEmp	id_people	CURRENT		91318569	A temporary employee, not on payroll – JJ Young, Manpower, etc.
500	Emeritus Faculty	DirAux	Dir_Aux_Ent	MR6190	9		School of Engineering Emeritus Faculty
500	Emeritus Faculty	DirAux	Dir_Aux_Ent	MR6210	9		Emeritus Faculty from the School of Science
500	Emeritus Faculty	DirAux	Dir_Aux_Ent	MR6200	9		Emeritus Faculty in Architecture
300	Vendor	Vendor	id_people	CURRENT		91067997	A vendor
300	Retiree	Retiree	id_people	CURRENT		91080042	A retiree
300	Vendor	N/A	id_people	CURRENT		91398702	A person affiliated with an on campus vendor
300	Vendor	N/A	id_people	CURRENT		91402506	A vendor working with a specific department on campus.
300	Vendor	N/A	id_people	CURRENT		91393994	Somone affiliated with an off campus vendor
300	Tech Park	TechPark	id_people	CURRENT		91080046	An employee at a tech park company
260	RU Club Member	N/A	id_people	CURRENT		91405098	A member of a Rensselaer Union sponsored club or organization.
250	Special Programs	Spec Prg	id_people	CURRENT		91121096	Special Program
200	Special Access	Spec Acc	id_people	CURRENT		91068269	Special Access Card
200	Conference Card	ConfCard	id_people	CURRENT		91114523	A Generic Conference ID card.
150	Residence Hall Occupant	ResLife	banner_students				A person with an active housing contract. May not be a student nor employee.
55	Retired Faculty	DirAux	Dir_Aux_Ent	S6160	9		A retired faculty member. This status is maintained by the Provost's office.

THE USENIX ASSOCIATION

Since 1975, the USENIX Association has brought together the community of system administrators, developers, programmers, and engineers working on the cutting edge of the computing world. USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems. USENIX and its members are dedicated to:

- problem-solving with a practical bias
- fostering technical excellence and innovation
- encouraging computing outreach in the community at large
- providing a neutral forum for the discussion of critical issues

Membership Benefits

- Free subscription to *;login:*, the Association's magazine, both in print and online
- Online access to all Conference Proceedings from 1993 to the present
- Access to the USENIX Jobs Board: Perfect for those who are looking for work or are looking to hire from the talented pool of USENIX members
- The right to vote in USENIX Association elections
- Discounts on technical sessions registration fees for all USENIX-sponsored and co-sponsored events
- Discounts on purchasing printed Proceedings, CD-ROMs, and other Association publications
- Discounts on industry-related publications; see http://www.usenix.org/membership/specialdisc.html

SAGE, The System Administrators Guild

SAGE is a Special Interest Group (SIG) of the USENIX Association. It is organized to advance the status of computer system administration as a profession, establish standards of professional excellence and recognize those who attain them, develop guidelines for improving the technical and managerial capabilities of members of the profession, and promote activities that advance the state of the art or the community.

For more information about membership, conferences, or publications, see http://www.usenix.org.

USENIX & SAGE Thank Their Supporting Members

USENIX Supporting Members

Addison-Wesley Professional/Prentice Hall Professional • AMD • Asian Development Bank
Cambridge Computer Services, Inc. • EAGLE Software, Inc. • Electronic Frontier Foundation
Eli Research • GroundWork Open Source Solutions • Hewlett-Packard • IBM • Intel • Interhack
The Measurement Factory • Microsoft Research • NetApp • Oracle • OSDL • Perfect Order
Raytheon • Ripe NCC • Sendmail, Inc. • Splunk • Sun Microsystems, Inc.
Taos • Tellme Networks • UUNET Technologies, Inc.

SAGE Supporting Members

Asian Development Bank • FOTO SEARCH Stock Footage and Stock Photography
Microsoft Research • MSB Associates • Raytheon • Splunk • Taos • Tellme Networks

ISBN 1-931971-38-2